物联网工程与技术规划教材

射频识别（RFID）协议原理及实践开发

邓　昀　李小龙　张　澎　主　编
陶晓玲　程小辉　王鲁达　编　著

電子工業出版社
Publishing House of Electronics Industry
北京・BEIJING

内 容 简 介

本书根据应用型本科院校物联网工程专业的 RFID 应用技术及相关课程的理论、实验教学要求编写。本书的主要内容包括：RFID 技术及基础知识介绍，RFID 高频 ISO 15693 协议原理及实践开发，RFID 高频 ISO 14443 协议原理及实践开发，RFID 超高频 ISO 18000-6 协议原理及实践开发，2.4G 有源 RFID 协议原理及实践开发，RFID 低频 125K ID 卡协议原理及实践开发，以及 RFID 与 Zigbee 相结合的综合应用开发。本书的所有开发例程都是在基于工程应用背景的开放实验平台上完成的，有效地将 RFID 技术与物联网工程应用相结合，有利于培养读者的工程应用能力。

本书可以作为物联网工程专业本科学生及物联网技术专业高职学生的 RFID 应用技术及相关课程的理论、实验教学指导，也可供从事物联网相关技术工作的技术人员参考。

图书在版编目（CIP）数据

射频识别（RFID）协议原理及实践开发 / 邓昀，李小龙，张澎主编. —北京：电子工业出版社，2016.6
ISBN 978-7-121-29070-1

Ⅰ. ①射… Ⅱ. ①邓… ②李… ③张… Ⅲ. ①无线射频识别－通信协议－高等学校－教材
Ⅳ. ①TN911.23

中国版本图书馆 CIP 数据核字（2016）第 132523 号

策划编辑：冉 哲
责任编辑：郝黎明
印 刷：
装 订：
出版发行：电子工业出版社
北京市海淀区万寿路 173 信箱 邮编 100036
开 本：787×1 092 1/16 印张：11.25 字数：281.6 千字
版 次：2016 年 6 月第 1 版
印 次：2016 年 6 月第 1 次印刷
定 价：29.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：ran@phei.com.cn。

前　言

本书根据应用型本科院校物联网工程专业的 RFID 应用技术及相关课程的理论、实验教学要求编写。

随着物联网技术在各领域应用的不断深入，对物联网工程专业应用型人才的培养提出了新的更高的要求。对物联网工程专业本科学生的培养，不仅要求掌握物联网技术的理论知识，还要求具备扎实的物联网系统分析、设计、开发能力。RFID 是物联网应用系统中的重要感知技术之一，RFID 应用技术是构建物联网应用系统的关键技术之一，物联网工程专业学生学习掌握 RFID 应用技术是以后从事物联网技术工作的基础。RFID 应用技术课程的应用性较强，对学生在 RFID 技术实际工程应用能力上的培养是一个需要不断探索改进的课题，本书作者及专业教学团队在编写本书时，试图以工程应用为背景，构建实用的 RFID 协议原理及实践开发教学体系，通过实践教学，培养学生在物联网工程应用中的 RFID 技术设计开发能力。

作者围绕应用型本科院校物联网工程专业人才培养的目标，联合广西、湖南及广东部分应用型本科院校的物联网工程专业教学团队，就如何培养学生的物联网应用系统设计开发能力，开展了长期的探讨与实验教学改革实践。通过长期的探讨与实践，我们认为，要真正提高学生的工程技术应用能力，需要解决好以下几个方面的问题。

（1）提高实践教学环节在人才培养方案中的比重。

（2）要建设以工程应用为背景、开放并可定制的实验实训平台。

（3）开发与实验实训平台紧密结合的实践性教材。

（4）在以工程应用为背景、开放的实验实训平台的建设及实践教材的开发编写过程中进一步提高专业教学团队的实践教学执行能力。

本书正是作者及专业教学团队在以工程应用为背景、开放的 RFID 开发平台建设的基础上编写的，因此本书具有较强的应用性及实用性。

本书共分 8 章，第 1 章和第 2 章分别介绍 RFID 技术及基础知识，这是后续各章节开展相关实践开发的基础。第 3～7 章分别以实际工程中的应用为背景，对在物联网工程中广泛应用的 RFID 模块的相关协议的工作原理、编程应用的方法和过程给予了详细的介绍与指导，其中第 3 章介绍 RFID 高频 ISO 15693 协议原理及实践开发，第 4 章介绍 RFID 高频 ISO 14443 协议原理及实践开发，第 5 章介绍 RFID 超高频 ISO 18000-6 协议原理及实践开发，第 6 章介绍 2.4G 有源 RFID 协议原理及实践开发，第 7 章介绍 RFID 低频 125K ID 卡协议原理及实践开发。第 8 章介绍 RFID 与 ZigBee 互联原理及实践开发，以提升学生的综合应用能力。书中包含的所有 RFID 模块的硬件原理图、实例源代码都可以通过华信教育资源网（www.hxedu.com.cn）免费注册后下载。

本书可以作为物联网工程专业本科学生及物联网技术专业高职学生的 RFID 应用技术及相关课程的理论、实验教学指导，也适用于从事物联网相关技术工作的技术人员参考。在应用型本科院校物联网工程专业的 RFID 应用技术课程的教学中，建议教学课时为 22～28 学时。

本书由邓昀教授、李小龙教授、张彭老师主编，参与编写的还有陶晓玲、程小辉、王鲁达。

本书是物联网工程规划教材，本书的出版得到了桂林理工大学、桂林电子科技大学、桂林市华智信息科技有限公司的支持，得到了国家自然科学基金项目（61462021）的资助。

由于作者水平有限，加之时间紧迫，可能存在错漏之处，恳请读者批评指正。

邓　昀

于桂林理工大学

目　录

第 1 章　RFID 技术简介

1.1　RFID 定义

无线射频识别技术（Radio Frequency Identification，RFID）是一种非接触的自动识别技术，它一般由标签（Tag）、阅读器（Reader）和天线（Antenna）三部分组成。它通过射频信号自动识别目标对象并获取相关数据，识别工作无须人工干预，可工作于各种恶劣环境。RFID 技术可识别高速运动物体并可同时识别多个识别卡，操作快捷方便。

RFID 射频识别是一种世界上较为领先的技术：第一，可以识别单个的非常具体的物体，而不是像条形码那样只能识别一类物体；第二，其采用无线电射频，可以透过外部材料读取数据，而条形码必须靠激光来读取信息；第三，可以同时对多个物体进行读取，而条形码只能一个一个地读取。此外，存储的信息量也非常大。

1.2　RFID 的发展历史

RFID 的出现可追溯至 20 世纪 30 年代，当然其基本技术无线电射频技术还可以追溯至 1897 年 Guglielmo Marconi 发明无线电的时候。RFID 采用与无线电广播相同的物理原理来发射和接收数据。

RFID 的基本前端系统一般由三部分组成：标签（Tag）或者雷达收发器（Transponder）、接收器（Receiver）、阅读器（Reader）天线。而这些部件则有许多变体，基于不同的功率、发射范围和距离、天线设计、工作频率、数据容量、管理和操作软件、数据编码格式、空中接口和通信协议等。这样，便出现了许多不同类型的系统，具有不同的特点和针对的应用范畴。

这些应用中涉及和影响当今社会、生活、经济、军事、法律和文化的各个方面。而目前最热烈和最受关注的莫过于廉价标签在商品（货物）流通生命周期过程中的识别应用。

1.2.1　发展和趋势

RFID 技术很早就和军事联系在一起。在 20 世纪 30 年代，美国陆军和海军都面临着在陆地、海上和空中对目标识别的问题。1937 年，美国海军研究试验室（U.S. Naval Research Laboratory，NRL）开发了敌我识别系统（Identification Friend-or-Foe（IFF）System），来将盟军的飞机和敌方的飞机区别开来。这种技术后来在 20 世纪 50 年代成为现代空中交通管制的基础。并且是早期 RFID 技术的萌芽，而优先应用在军事、实验室等。

早期系统组件昂贵而庞大，但随着集成电路、可编程存储器、微处理器及软件技术和编程语言的发展，创造了 RFID 技术推广和部署的基础。

20 世纪 60 年代后期和 70 年代早期，有些公司（如 Sensormatic 和 Checkpoint Systems）开始推广相对比较简单的 RFID 系统的商用产品，主要用于电子物品监控（Electronic Article Surveillance，EAS），即保证仓库、图书馆等物品的安全和监视。这种早期的商业 RFID 系统，称为 1 比特标签系统，相对容易构建、部署和维护。但是这种 1 比特系统只能检测被表示的目标是否在场，不能有更大的数据容量，甚至不能区分被标识目标之间的差别。

早期的 RFID 发展里程碑如图 1-1 所示。

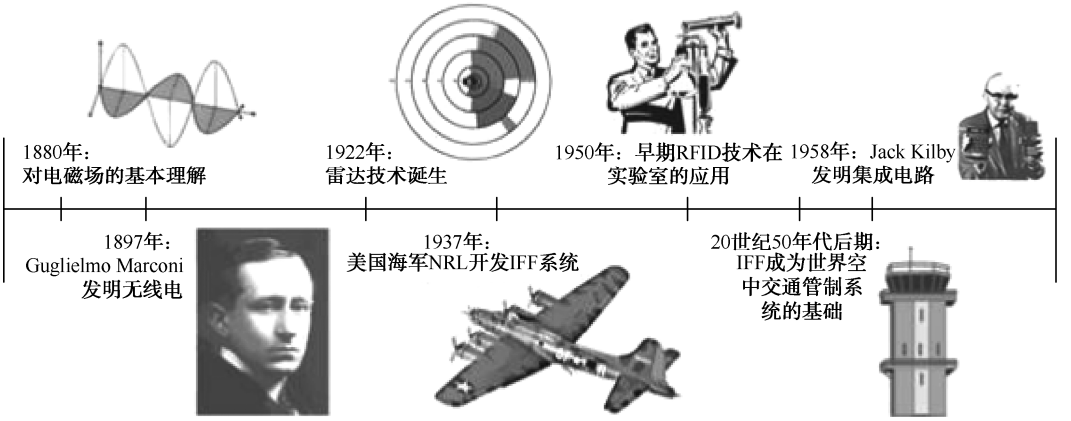


图 1-1　早期的 RFID 发展里程碑

在 20 世纪 70 年代，制造、运输、仓储等行业都试图研究和开发基于 IC 的 RFID 系统的应用，如工业自动化、动物识别、车辆跟踪等。在此期间，基于 IC 的标签体现出了可读写存储器、更快的速度、更远的距离等优点。但这些早期的系统仍然是专有的设计、没有相关标准、也没有功率和频率的管理。

在 20 世纪 80 年代早期，出现更加完善的 RFID 技术和应用，如铁路车辆的识别、农场动物和农产品的跟踪。

20 世纪 90 年代，道路电子收费系统在大西洋沿岸得到广泛应用，从意大利、法国、西班牙、葡萄牙、挪威，到美国的达拉斯、纽约和新泽西。这些系统提供了更完善的访问控制特征，因为它们集成了支付功能，也成为综合性的集成 RFID 应用的开始。

从 20 世纪 90 年代开始，多个区域和公司开始注意这些系统之间的互操作性，即运行频率和通信协议的标准化问题。只有标准化，才能将 RFID 的自动识别技术得到更广泛的应用，如这时期美国出现的 E-ZPass 系统。

同时，作为访问控制和物理安全的手段，RFID 卡钥匙开始流行起来，试图取代传统的访问控制机制。这种称为非接触式的 IC 智能卡具有较强的数据存储和处理能力，能够针对持有人进行个性化处理，也能够更灵活地实现访问控制策略。唯一性识别的应用如图 1-2 所示。

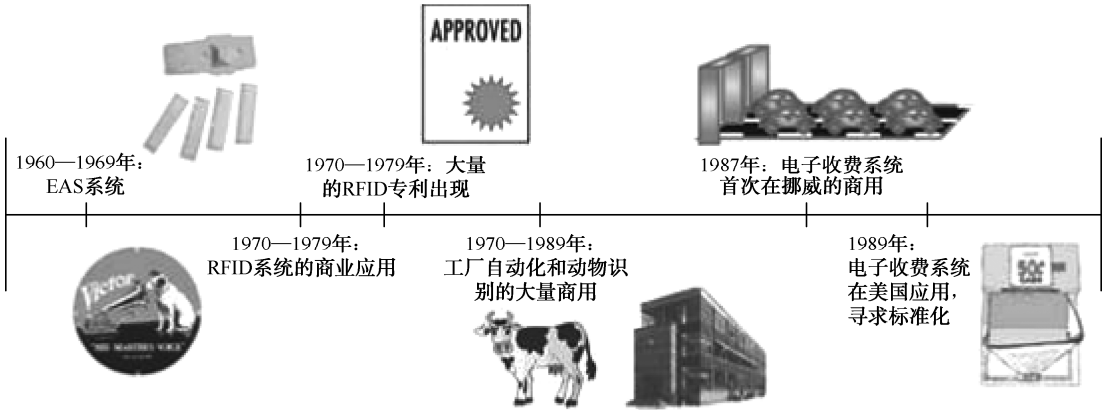


图 1-2　唯一性识别的应用

1.2.2 RFID 的热潮和整合性应用

在 20 世纪末期，大量的 RFID 应用指数般地试图扩展到全球范围。

在美国，Texas Instruments 则是这方面的推动先锋。TI 从 1991 年开始建立德州仪器注册和识别系统（Texas Instruments Registration and Identification Systems，TIRIS）。该系统如今称为 TI-RFid（Texas Instruments Radio Frequency Identification System），已经是一个主要的 RFID 应用开发平台。

在欧洲，EM Microelectronic-Marin 从 1971 年开始研究超低功率的集成电路。1982 年，Mikron Integrated Microelectronics 开始了 ASIC 技术，并在 1987 年由其奥地利分公司开始开发识别和智能卡芯片。1995 年，Philips Semiconductors 收购了 Mikron Graz，如今 EM Microelectronic 和 Philips Semiconductors 是欧洲的主要 RFID 厂商。

从技术上看，数年前，所部署的 RFID 应用基本上都是低频（LF）和高频（HF）的被动式 RFID 技术。LF 和 HF 系统都具有优先的数据传输速度和有效距离。因此，有效距离限制了可部署性。数据传输速度则限制了其可伸缩性。因此，20 世纪 90 年代后期，开始出现超高频（UHF）的主动式标签技术，提供更远的传输距离，更快的传输速度。基于此，重载的企业应用才开始使用这种技术，如供应链管理中的托盘和包装跟踪、存货和仓库管理、集装箱管理、物流管理等。并且逐渐试图成为合成的企业应用（包括 ERP、SCM、CRM、EAM、B2B 等）的数据和语义基础。

从 20 世纪 90 年代末期到现在，零售巨头（如 Wal-Mart、Target、Metro Group）及一些政府机构［(如美国国防部（DoD)］都开始推进 RFID 应用，并要求他们的供应商也采用此技术。同时，标准化的纷争出现了多个全球性的 RFID 标准和技术联盟，主要有 EPCglobal、AIM Global、ISO/IEC、UID、IP-X 等。这些组织主要在标签技术、频率、数据标准、传输和接口协议、网络运营和管理、行业应用等方面试图达成全球统一的平台，如图 1-3 所示。

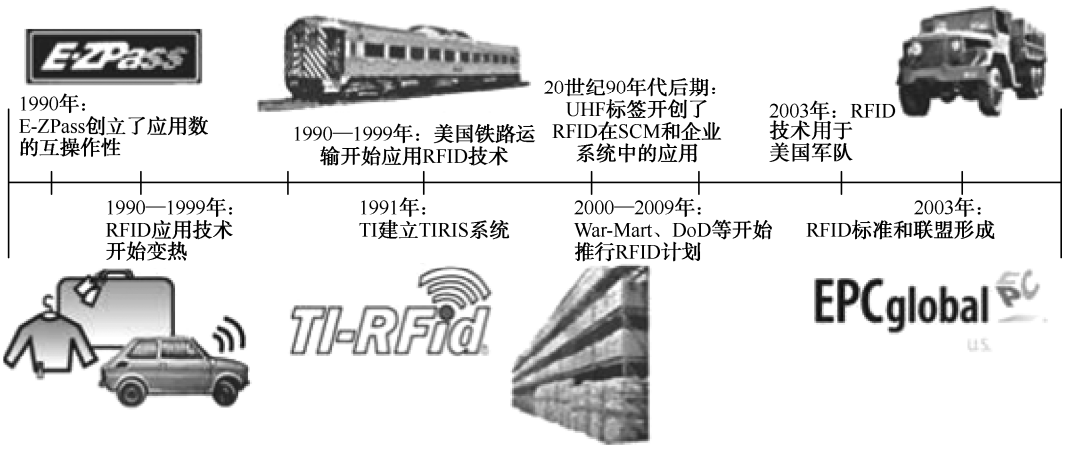


图 1-3　整合应用开始

1.3 RFID 系统的组成

RFID 系统通常由以下两个组件组成，如图 1-4 所示。

（1）收发器（Transponder）：位于被识别的对象。

（2）讯问器（Interrogator）或者阅读器（Reader）：取决于设计和所采用的技术，可以是阅读或者读写设备。

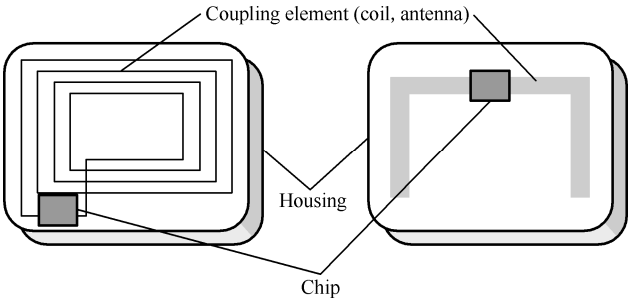


图 1-4　RIFD 数据承载设备布局

阅读器通常包含一个射频模块（发射器和接收器）、一个控制单元和一个与收发器的耦合单元。另外，某些阅读器还包含其他数据接口系统（RS-232、RS-485、TCP/IP 等），以便将数据转发到其他系统（PC、机器人控制系统等）。

雷达收发器：表示 RFID 系统的实际数据载体，通常由一个耦合单元和一个电子芯片组成。雷达收发器通常不具备自身电源供应，当它不在质询器的质询范围时，整体呈被动状态。它只有在质询器的质询范围之内才被激活。激活雷达收发器的电力通过耦合单元传输给收发器，所需的数据和时钟脉冲也是如此。图 1-4 为 RFID 数据承载设备的主要布局，左边是具有天线线圈的感应耦合 Transponder；右边是具有偶极天线的微波 Tag/Transponder。

1.4　RFID 的应用

1．低频（工作频段为 125kHz～134kHz）

RFID 技术首先在无源低频得到广泛的应用和推广。其原理是通过读写器线圈和识别卡内部线圈间的电磁耦合的方式进行工作,读写器的交变电磁场在识别卡天线中感应出电压，被整流后作为识别卡供电电压使用。这种方法虽然电磁场的区域能够很好地被定义，但是磁场强度下降得太快。就频率而言，低频 RFID 具有以下特性。

（1）一般低频的工作频段为 120kHz～134kHz，通常采用频率 134.2kHz，该频段的波长大约为 2500m。

（2）除了金属材料影响外，一般低频能够穿过任意材料的物品而不降低它的读取距离。

（3）工作在低频的读写器在全球没有任何特殊的许可限制。

（4）低频产品有不同的封装形式。好的封装形式使用寿命可达 10 年左右，但是价格高。

（5）虽然低频的磁场区域下降快，但是能够产生相对均匀的读写区域。

（6）相对于其他频段数据传输速率比较慢。

（7）读卡器的价格相对于其他频段来说要贵些。

低频 RFID 主要应用于以下几个方面。

① 畜牧业的管理系统。

② 汽车防盗和无钥匙开门系统的应用。

③ 马拉松赛跑系统的应用。

④ 自动停车场收费和车辆管理系统。

⑤ 自动加油系统的应用。
⑥ 酒店门锁系统的应用。
⑦ 门禁和安全管理系统。

2．高频（工作频率主要为 13.56MHz）

高频下识别卡不需要绕制线圈，可以通过印刷的方式制作天线。识别卡一般是负载调制的方式工作，也就是通过识别卡的负载电阻的接通和断开促使读写器天线上的电压发生变化，实现远距离识别卡对天线电压进行振幅调制。就频率而言，高频 RFID 具有以下特性。

（1）工作频率为 13.56MHz，该频率的波长大概为 22m。

（2）除了金属材料外，该频率的波长可以穿过大多数的材料，但是往往会降低读取距离。识别卡（感应器）需要离开金属一段距离。

（3）该频段在全球都得到认可并没有特殊的限制。

（4）该系统具有防冲撞特性，可以同时读取多个识别卡。

（5）可以把某些数据信息写入识别卡中。

（6）数据传输速率比低频要快且价格不是很贵。

高频 RFID 主要应用于以下几个方面。

① 图书管理系统的应用。
② 瓦斯钢瓶的管理应用。
③ 服装生产线和物流系统的管理和应用。
④ 预收费系统。
⑤ 酒店门锁的管理和应用。
⑥ 大型会议人员通道系统。
⑦ 固定资产的管理系统。
⑧ 医药物流系统的管理和应用。
⑨ 智能货架的管理。

3．超高频（工作频段为 860MHz～960MHz）

超高频系统通过电场来传输能量。电场的能量下降的不是很快，但是读取的区域不能很好进行定义。该频段读取距离比较远，无源可达 10m 左右，主要是通过电容耦合的方式进行实现。超高频 RFID 具有以下特性。

（1）在该频段，全球的定义不太相同，欧洲和部分亚洲定义的频率为 868MHz，北美定义的频段为 902MHz～905MHz，日本定义的频段为 950MHz～956MHz。该频段的波长大概为 30cm。

（2）目前该频段 RFID 输出的功率在美国定为 4W，欧洲定为 500mW。

（3）超高频段的电波容易受到水、灰尘、雾等悬浮颗粒的散射，因此许多材料都不易通过。相对于高频的识别卡来说，超高频段的识别卡不需要和金属分开。

（4）超高频段识别卡的天线一般是长条或标签状。天线有线性和圆极化两种设计，以满足不同应用的需求。

（5）超高频段有较好的读取距离，但是对读取区域很难进行定义。

（6）有很高的数据传输速率，在很短的时间可以读取大量的识别卡。

超高频 RFID 主要应用于以下几个方面。

① 供应链的管理和应用。
② 生产线自动化的管理和应用。
③ 航空包裹的管理和应用。
④ 集装箱的管理和应用。
⑤ 铁路包裹的管理和应用。
⑥ 后勤管理系统的应用。

第 2 章　RFID 相关基础知识

2.1　工作频率及应用范围

目前定义 RFID 产品的工作频率主要有 3 个频段：低频、高频、超高频。

1．低频（LF 125kHz～134kHz）

此频段的感应器（标签）形状多样，无法制作成标签形式，因此完全不适合 RFID 条码打印机，可不讨论。

2．高频（HF 13.56MHz）

高频由于可通过腐蚀印刷的方式制作天线，其标签可制成硬卡片状或标签状。此频段主要应用于图书管理系统、瓦斯钢瓶管理、服装生产线和物流系统、预收费系统、酒店门锁的管理、大型会议人员通道系统、固定资产管理系统、医药物流系统管理、智能货架管理、身份识别（身份证）。

3．超高频（UHF 860MHz～960MHz）

超高频系统通过电场来传输能量，主要是通过电磁耦合的方式进行实现。读取距离比较远，无源可达 10m 左右，电子标签的天线一般是长条和标签状（智能标签）。此频段主要应用于供应链管理、生产线自动化管理、航空包裹管理、集装箱管理、铁路包裹管理、后勤管理系统。根据 RFID 打印机的特殊要求，高频段的标签状电子标签及超高频段的标签，既可以打印条码又可以进行电子编码，此种类型的智能标签适合在 RFID 打印机打印。所以这两种标签的相关应用应是今后关注的重点。

2.2　相关协议标准

1．高频段（HF）射频标签相关的国际标准

高频段射频标签目前具有全球统一 13.56MHz 的工作频率。该频段的射频标签称为高频标签。是目前实际应用最多且技术最成熟的射频标签技术。符合国际标准的有 ISO/IEC 14443 A/B、ISO/IEC 15693（兼容于 ISO/IEC 18000-3）、ISO/IEC 18000-3、EPC C1 HF、Ubiquitous ID。

其中，EPC C1 HF 兼容于 ISO/IEC 15693，而 ISO/IEC15693 兼容于 ISO/IEC18000-3 的 mode 1。

2．超高频（UHF）标签

超高频的射频标签的典型工作率为 860MHz～960MHz。对于该频段，全球的定义并不统一。欧洲和部分亚洲定义的频率为 868MHz，北美定义的频段为 902MHz～905MHz，日本定义的频段为 950MHz～956MHz。相应的国际标准有以下几种。

① ISO/IEC 18000-6A、B、C。

② EPC global C0、C1G1、C1G2，现已并入国际标准 ISO/IEC 18000，成为 ISO/IEC 18000-6C。

③ Ubiquitous ID 日本的组织，定义了 UID 编码结构和通信管理协议。

其他的还有 IPICO 公司的 IP-X 协议。

2.3 RFID 射频识别卡的分类

RFID 射频卡（以下简称射频卡）按不同的方式分类有以下几种。

（1）按供电方式分为有源卡和无源卡。

有源卡是指卡内有电池提供电源，其作用距离较远，且识别运动目标能力强，但需要定期更换电池，由于近年工艺改进延长了电池的使用寿命，可达 5 年以上。无源卡内无电池，它利用波束供电技术将接收到的射频能量转化为直流电源为卡内电路供电，但其作用距离近，识别运动目标能力低于有源卡。

（2）按载波频率分为低频射频卡、中频射频卡、高频射频卡。

低频射频卡主要有 125kHz 和 134.2kHz 两种，低频系统主要用于短距离、低成本的应用中，如多数的门禁控制、校园卡、动物监管、货物跟踪等。中频射频卡主要为 13.56MHz，中频系统用于门禁控制和需传送大量数据的应用系统。高频射频卡主要为 433MHz、915MHz、2.45GHz（微波）、5.8GHz（微波）等。高频系统应用于需要较长的读写距离和高读写速度的场合，其天线波束方向较窄且价格高，在火车监控、高速公路收费等系统中应用。

（3）按调制方式的不同可分为主动式和被动式。

主动式射频卡用自身的射频能量主动地发送数据给读写器，射频卡发射的信号仅穿过障碍物一次，因此主动方式工作的射频卡主要用于有障碍物的应用中，距离比被动式远，可达 30m。被动式射频卡使用调制散射方式发送数据，它必须利用读写器的载波来调制自己的信号，该类技术适用于门禁或交通应用中，因为读写器可以确保只激活一定范围之内的射频卡。在有障碍物的情况下，用调制散射方式，读写器的能量必须穿过障碍物两次。

（4）按作用距离可分为密耦合卡（作用距离小于 1cm）、近耦合卡（作用距离小于 15cm）、疏耦合卡（作用距离小于 1m）和远距离卡（作用距离从 1m 到 10m 甚至更远）。

（5）按芯片分为只读卡、读写卡和 CPU 卡。

2.4 RFID 技术的基本工作原理

下面按有源式和无源式两种情况叙述。

（1）有源 RFID 识别卡：内有电池提供电源，其作用距离较远，且识别运动目标能力强。有源识别卡又分为主动式和被动式，有源主动式是主动发送某一频率的信号，读写器读取信息，解码后送至中央信息系统进行有关数据处理。有源被动式是当识别卡进入读写器的电磁场范围时才被“唤醒”后发送出某一频率的信号。

（2）无源 RFID 识别卡：本身不带电池，依靠读卡器发送的电磁能量工作。当识别卡进入读写器天线磁场后，接收到读写器发出的射频信号，便获得感生电流，这时识别卡把存储在芯片中的产品信息发送出去。

2.5 RFID 相关术语和定义

1．术语和定义

（1）防冲突环（Anticollision Loop）：使用算法准备和处理能量所及范围内，一个 VCD 和多个 VICC 间的对话。

（2）字节（Byte）：由指定的 8 位数据 b1～b8 组成，从最高有效位（MSB，b8）到最低有效位（LSB，b1）。

2．符号 f_c

f_c：工作场频率（载波频率）。

3．缩略语

（1）AFI（Application Family Identifier）：应用族识别符，应用的卡预选准则。
（2）CRC（Cyclic Redundancy Check）：循环冗余校验。
（3）DSFID（Data Storage Format Identifier）：数据存储格式。
（4）EOF（End of Frame）：帧结束。
（5）LSB（Least Significant Bit）：最低有效位。
（6）MSB（Most Significant Bit）：最高有效位。
（7）RFU（Reserved for Future Use）：留作将来 ISO/IEC 使用。
（8）SOF（Start of Frame）：帧的起始。
（9）UID（Unique Identifier）：唯一序列号。
（10）VCD（Vicinity Coupling Device）：附近式耦合设备。
（11）VICC（Vicinity Integrated Circuit Card）：集成电路附近式卡。

第 3 章　RFID 高频 ISO 15693 协议原理及实践开发

本章所述内容所使用的设备包括桂林华智 RFID 物联网教学科研平台实验箱高频 ISO 15693 模块读写器、ISO 15693 卡片、串口线。主要操作内容：使用上位机软件，与 RFID 高频 ISO 15693 模块进行通信，完成对高频 ISO 15693 标签的操作。通过本章的学习，掌握对高频 ISO 15693 模块读写器的操作，了解相关协议及工作原理。在设计部分主要介绍的方面有应用程序开发所使用的环境、语言、原理及实现等，具体通过几个实验的操作及源码的分析使读者对 Windows 应用程序开发有一个了解，并能够设计一个可连接实验系统平台进行相关操作的应用程序。本章针对 RFID 实验平台的 ISO 15693 模块，包含 5 个设计实验：① 应用程序的建立及寻卡功能的实现；② 多数据块读写功能的实现；③ 应用族标识写入及锁定功能的实现；④ 数据存储格式标识写入及锁定功能的实现；⑤ 测试防冲突功能的实现。通过完成这些程序设计来掌握 Windows 窗体应用程序开发，同时对串口通信及相关协议有更深刻的理解。

1．读写 IOS 15693 协议读写器

读写 IOS 15693 协议读写器工作电压 5V，工作温度 0℃～60℃，工作频段是 HF 13.56MHz。ISO 15693 协议的每个电子标签都有一个 8 字节共 64 位的唯一序列号（UID），这个 UID 一方面可以使全球范围内的标签互相区别，更重要的是可以在多标签同时读/写时用于防冲突。8 字节的 UID 按权重从高到低标记为 UID7～UID0，其中 UID7 固定为十六进制数的 E0H，UID6 是标签制造商的代码，如 NXP 的代码为 04H，TI 的代码为 07H；UID5 为产品类别代码，如 ICODE SL2 ICS20 是 01H，Tag-it HF-I Plus Chip 为 80H，Tag-it HF-I Plus Inlay 为 00H。剩下的 UID4～UID0 为制造商内部分配的号码。

2．ISO 15693 协议电子标签

ISO 15693 协议电子标签数量众多，应用范围极为广泛。为了区分不同行业中的电子标签，ISO 用一个字节的 AFI（Application Family Identifier）来区分不同行业中的电子标签。ISO 15693 国际标准还规定了一个字节的可选的数据存储格式识别符（DSFID），用来区分标签中不同的数据存储格式。

3．标签状态

Power off 状态：在标签未进入到有效磁场区域时标签处于 Power off 状态。

Ready 状态：被激活后选择标识符未设立时，处理任何的请求。

Quiet 状态：不处理任何标签清点指令，可接收直接寻址的命令。

Select 状态：仅响应选择标识符设置的请求。

当阅读器辐射场的能量不能激活应答器，应答器处于 Power off 状态。当应答器被阅读器辐射场激活，所获能量可支持应答器正常工作时，应答器进入就绪（Ready）状态。

4．产品平面图

产品平面图如图 3-1 所示。

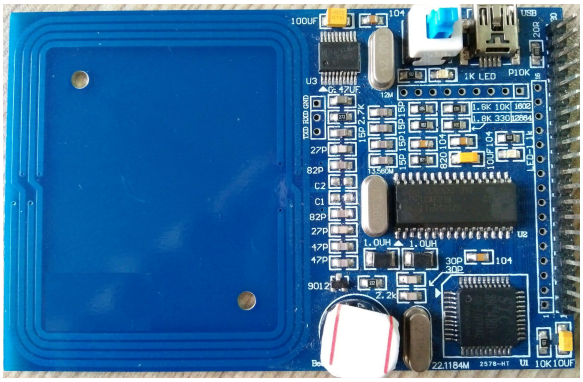


图 3-1　产品平面图

5．产品电路图

产品电路图如图 3-2 所示。

6．工作原理简介

MCU（微处理器 CPU）采用 AT89C54 芯片，驱动芯片为 CLRC632 芯片。

CLRC632 是恩智浦公司推出的适用于工作频率为 13.56MHz 的非接触式智能卡和标签射频基站芯片，并且支持这个频段范围内多种 ISO 非接触式标准，其中包括 ISO 14443 和 ISO 15693。CLRC632 特点如下。

工作电压为 3V～5V，读卡距离可达 10cm；标准并行接口与标准 SPI 接口；可读 ISO/IEC 14443 Type A 和 Type B 的卡；可读 ISO/IEC 15693 标准的卡。

CLRC632 负责读写器与非接触式智能卡和标签的读写等功能，其基本功能包括调制、解调、产生射频信号、安全管理和防冲突处理，是读写器 MCU（微控制器）与非接触式智能卡和标签交换信息的桥梁。

（1）CLRC632 硬件接口电路

MCU 的硬件内核接口电路可分为以下三个部分。

① 与 MCU（微处理器 CPU）接口电路。

② 与天线射频接口电路。

③ 与电源接口电路。

（2）CLRC632 的基本操作

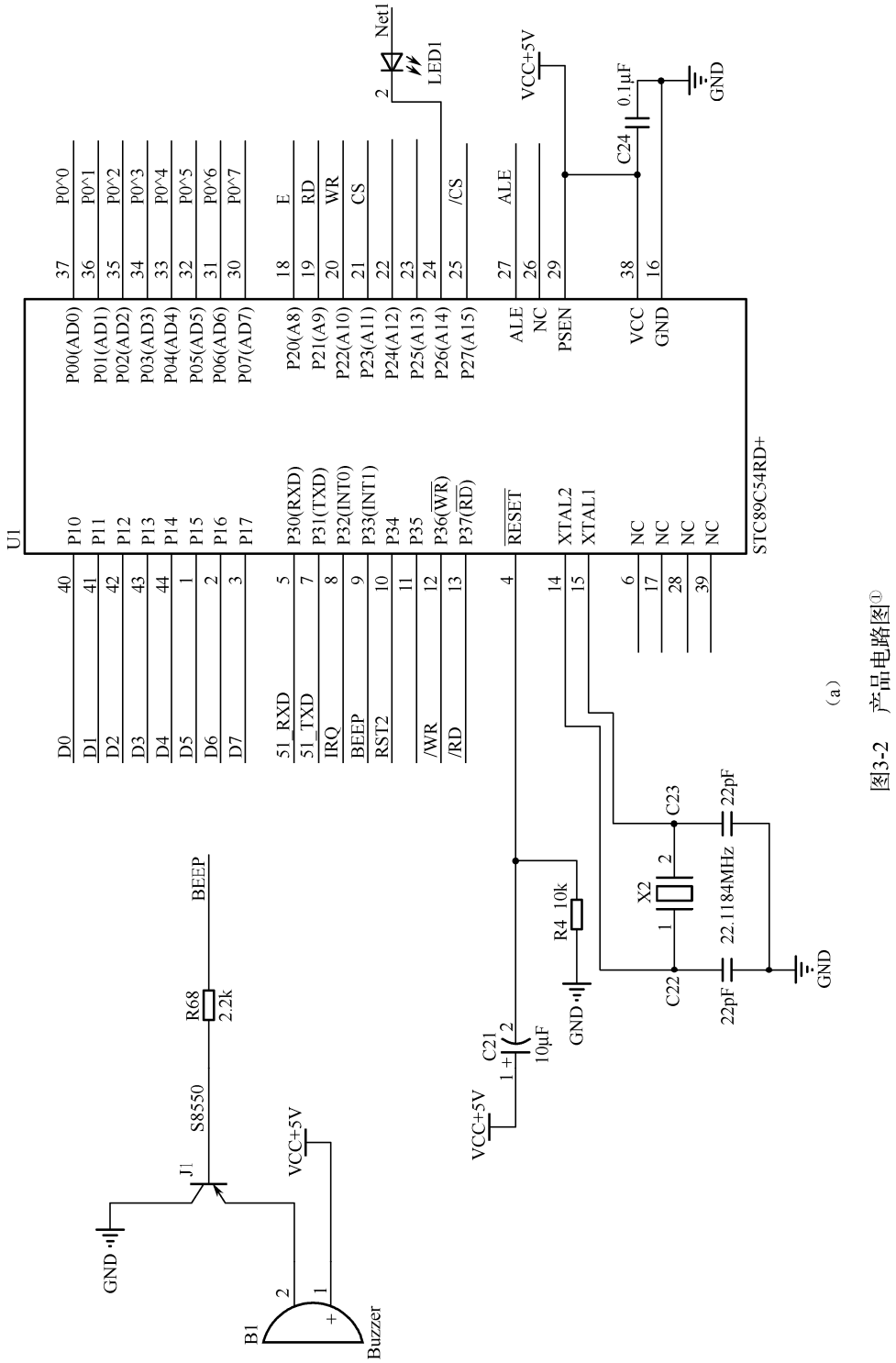
访问 CLRC632 的寄存器：MCU 通过对 CLRC632 的控制，实现非接触式智能卡和标签的读写操作。MCU 对 CLRC632 的控制有以下三种方式。

① 执行命令来初始化函数和控制数据操作。

② 通过设置配置位来设置电气和函数的行为。

③ 读状态标识 CLRC632 的状态。

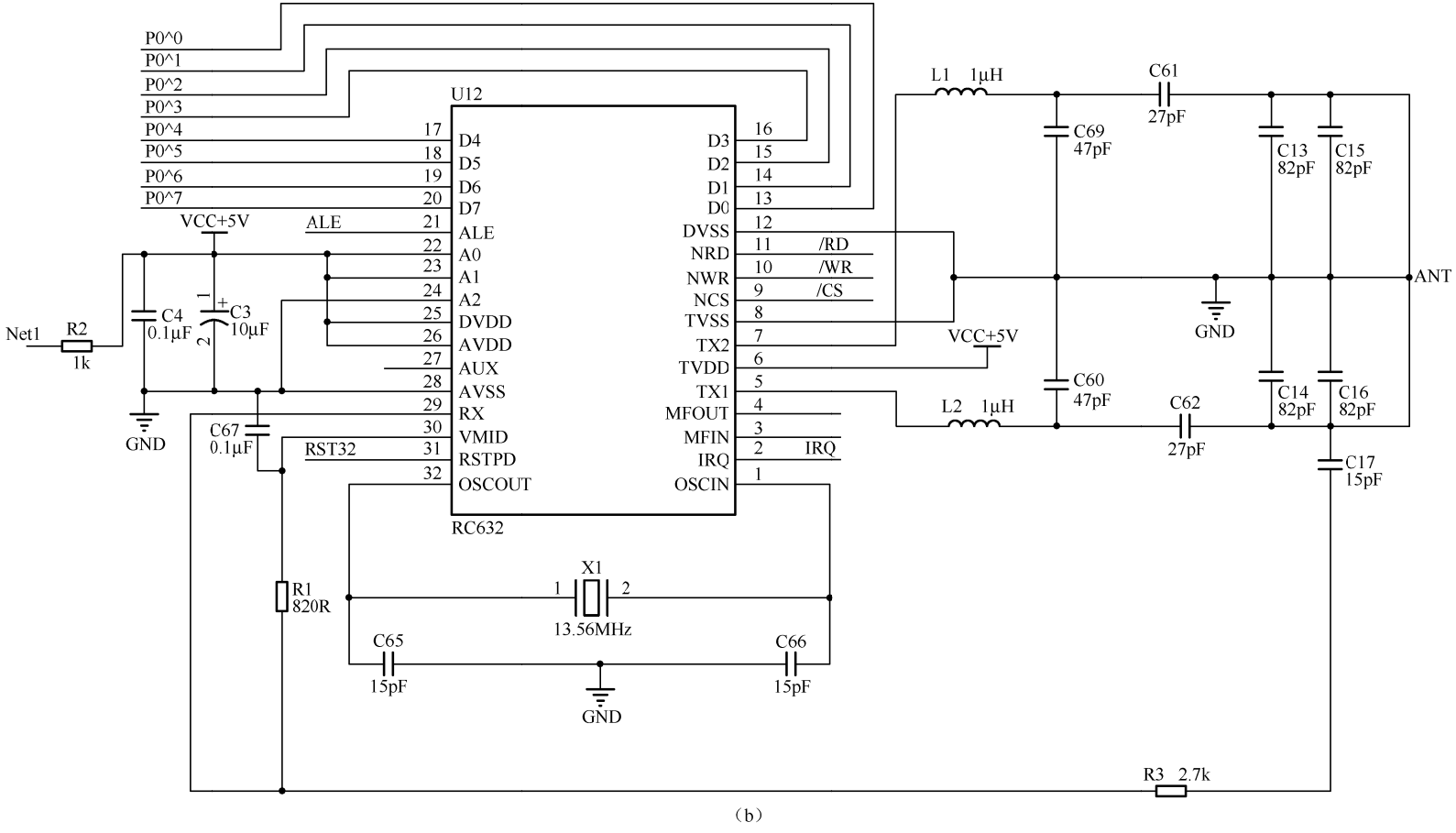
这三种方式本质都是通过读、写 CLRC632 的寄存器来实现。执行命令即是将命令代码写入 CLRC632 的命令寄存器，通过 CLRC632 的 FIFO 缓冲区来传递参数和交换数据；设置配置位即是设置 CLRC632 的寄存器的相应位；监控 CLRC632 的状态是通过读 CLRC632 的寄存器来实现的。CLRC632 内部有 64 个寄存器，这些寄存器被分为 8 页，每页有 8 个寄存器。无论是否被选中，页寄存器总是可以被访问。



（a）

图3-2　产品电路图①

①为了与软件保持一致，电路图中标识未进行修改。


P0^0
P0^1
P0^2
P0^3
P0^4
P0^5
P0^6
P0^7
VCC+5V
ALE
U12
D4
D5
D6
D7
ALE
A0
A1
A2
DVDD
AVDD
AUX
AVSS
RX
VMID
RSTPD
OSCOUT
D3
D2
D1
D0
DVSS
NRD
NWR
NCS
TVSS
TX2
TVDD
TX1
MFOUT
MFIN
IRQ
OSCIN
/RD
/WR
/CS
IRQ
RC632
L1 1μH
C61
27pF
C69
47pF
C13
82pF
C15
82pF
ANT
GND
VCC+5V
C60
47pF
C14
82pF
C16
82pF
L2 1μH
C62
27pF
C17
15pF
Net1
R2
1k
C4
0.1μF
C3
10μF
GND
C67
0.1μF
RST32
R1
820R
X1
13.56MHz
C65
15pF
C66
15pF
GND
R3 2.7k

（b）

图3.2　产品电路图（续）

（3）CLRC632 命令集

CLRC632 的行为是通过其内部状态机执行特定的命令来实现的。通过将命令代码写入命令寄存器来开始这些命令。执行命令所需要的参数和数据主要通过 MFRC500 内部的 FIFO 缓冲区来实现交换。

具体实现交换的步骤如下。

① 如果某条命令执行时，需要输入数据，它会将 FIFO 缓冲区中找到的数据作为输入数据。

② 如果某条命令执行时，需要几个参数，只有通过 FIFO 缓冲区得到正确数据的参数后，该命令才可以开始。

③ 在命令开始时，FIFO 缓冲区并不自动清空。所以将命令参数和数据写入 FIFO 缓冲区后，开始命令。

④ 微处理器可以写入新的命令代码到命令寄存器来中断任何正在执行的命令，如 Idle 命令，但 StartUp 命令不可以被中断。

7．工作原理

读写器向标签发一组固定频率的电磁波，卡片内有一个 LC 串联谐振电路，其频率与读写器发射的频率相同，在电磁波的激励下，LC 谐振电路产生共振，从而使电容内有了电荷，在这个电容的另一端，接一个单向导通的电子泵，将电容内的电荷送到另一个电容内储存，当所积累的电荷达到 2V 时，此电容可作为电源，为其他电路提供工作电压，将卡内数据发射出去或接收读写器的数据。

3.1 HF13.56MHz ISO 15693 模块寻卡

3.1.1 协议原理

（1）ISO 15693 标准规范第三部分协议和指令内容。

（2）唯一序列号（UID）。VICC 由一个 64 位（bit）的 UID 唯一标识，如图 3-3 所示。在 VCD 和 VICC 之间防冲突和一对一交换期间，用来定位每个唯一的特别的 VICC。

MSB LSB

64 57	56 49	48 1
“E0”	IC Mfg code	IC制造商序列号

图 3-3 唯一序列号

由图 3-3 可知，UID 应该永久存在而且由 IC 制造商进行设定。

UID 主要包括以下内容。

① 8 位 MSB 应为“E0”。

② 根据 ISO/IEC 7816-6：1996/Amd.1，IC 制造商序列号 8 位。

③ 由制造商指定的 48 位唯一的流水号。

（3）RFID 系统发送到上位机软件的数据包就是一次响应，一次响应的数据包含在一帧内，帧分隔符（SOF，EOF）；一帧中传输位的个数是 8 的整数倍，即整数个字节。

一个单字节域在传输过程中首先传输最低有效位（LSBit）；而一个多字节域在传输过程中

首先传输最低有效字节（LSByte），每个字节首先传输最低有效位（LSBit）。每次请求包括的域有标志、强制和可选的参数域，取决于命令、应用数据域和 CRC 校验码。

上位机发送数据命令格式如下：

SOF	CMD	协议 Type	Select	DataLength	Data	保留	EOF
2B	1B	1B	1B	2B	DataLengthByte	2B	2B

① SOF：帧头 0xEE 0xCC。

② CMD：命令字节，如表 3-1 所示。

表 3-1　命令字节

CMD	描述	Data	
00	寻多卡	发送	无数据段
		回应	UID 的 8 字节的倍数
01	寻单卡	发送	无数据段
		回应	UID
02	保持静默	发送	UID
		回应	无数据段
03	选择	发送	UID
		回应	无数据段
04	重置到准备状态	发送	UID
		回应	无数据段
05	写入应用族标识	发送	UID+1B（族标识）
		回应	无数据段
06	锁应用族标识	发送	UID
		回应	无数据段
07	读多块	发送	UID+1B（起始块号）+1B（块数）
		回应	块数据（数据从起始块数据开始）
08	写多块	发送	UID+1B（起始块号）+1（块数）+数据
		回应	无数据段
09	锁块	发送	UID+1B（块号）
		回应	无数据段
0A	写存储格式标识	发送	UID+1B（存储格式）
		回应	无数据段
0B	锁存储格式标识	发送	UID
		回应	无数据段
0C	获取系统信息	发送	UID
		回应	1B（族标识）+1B（存储标识）+（块数）1B+1B（每块的字节数）

③ 协议 Type：01—ISO 15693；02—ISO 14443 协议；03—ISO 14443 高频协议。

④ Select：选择模式。0—不选择模式；1—选择模式（选择模式下发送帧可以不包含 UID，但是占有字节数）。

⑤ DataLength：Data 段的字节数。

⑥ 保留：2B 保留（扩展）。

⑦ EOF：0x0D 0x0A。

读写器返回数据命令格式如下：

SOF	Status	CMD	协议 Type	DataLength	Data	保留	EOF
2B	1B	1B	1B	2B	DataLengthByte	2B	2B

Status 描述，如表 3-2 所示。

表 3-2　Status 描述

Status	描述
0x00	命令执行成功
0x01	关闭串口失败
0x02	静默失败
0x0F	没有读到标签/卡
0x10	命令不支持
0x11	命令不被允许
0x12	硬件类型不匹配

其他字段描述与发送命令字段相同。

本小节内容涉及寻单卡与寻多卡操作。上位机执行寻卡命令向下位机发送数据命令：

```
EE CC 01 01 00 00 01 00 00 00 0D 0A（寻单卡）
EE CC 00 01 00 00 01 00 00 00 0D 0A（寻多卡）
```

其中，开始两个字节 0xEE 0xCC 为帧头，标识一帧数据的开始；最后两个字节 0x0D 0x0A 为帧尾，标识一帧数据结束；第三字节 0x01（寻单卡）0x00（寻多卡）为 CMD 字段，表示当前命令为寻单（多）卡命令；第四字节 0x01 为协议 Type 字段，表示当前协议为 ISO 15693；第五字节 0x00 为是否在选择模式字段，表示当前模式不在选择模式；第六、七字节 0x00 0x01 为 DataLength 字段，表示 Data 字段一个字节长度。第八字节 0x00 为 Data 字段；第九、十字节 00 00 为保留区（用于其他扩展）字段。

寻卡命令返回（以寻到的标签号为 0C24572A000104E0 为例）：命令解析参见读写器返回数据命令格式。

```
EE CC 00 01 01 00 08 0C 24 57 2A 00 01 04 E0 00 00 0D 0A（寻单卡返回）
EE CC 00 00 01 00 08 0C 24 57 2A 00 01 04 E0 00 00 0D 0A（寻多卡返回）
```

3.1.2　操作步骤

（1）打开上位机软件及 RFID 原理机的电源，使用串口线建立通信连接。

（2）串口打开。将卡片放到识别区（如果连接了外接天线，将卡片放置在天线旁边），在执行寻卡操作之前需要将串口（默认波特率为 115200，若改成其他值，可能造成无法正确接收到数据）打开，打开串口的操作步骤如图 3-4 所示。

图 3-4　打开串口

（3）单卡识别。打开串口成功后，自动跳到寻卡选项并在弹出寻卡操作界面，选中“单卡识别”单选按钮，单击“寻卡”按钮，上位机向下位机发送寻单卡命令，读写器执行寻卡命令，将数据返回给上位机，上位机接收数据线程（在打开串口成功后启动）接收数据并进行数据解析。寻卡成功返回如图 3-5 所示的信息。

图 3-5　寻卡操作

（4）多卡识别。选择“多卡识别”，单击“寻卡”按钮进行多卡识别。ISO 15693 电子标签的防冲突与 ISO 14443A 中基于位的防冲突类似。其最根本的一点就是基于标签有一个全球唯一的序列号。因为序列号的唯一性，所以全球范围内的任意两个标签，其 64 位的序列号中总有 1 位的值是不一样的，也就是说，任意两个标签的序列号总有一个某 1 位是“0”，另一

个是“1”。防冲突的过程可以 1 位 1 位地进行，也可以 4 位 4 位地进行。ISO 15693 采用位和时隙相结合的防冲突机制。

多卡识别寻成功返回如图 3-6 所示的信息。

图 3-6　多卡识别

（5）自动寻卡。自动寻卡即上位机程序使用线程不断地向读写器发送寻卡命令，上位机接收数据线程始终扫描读写器返回的串口数据进行数据获取。自动寻单卡成功返回如图 3-7 所示的信息。

图 3-7　自动寻卡

3.1.3 程序设计：应用程序的建立及寻卡功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境下创建 C#窗体应用程序；编写代码，通过上位机软件完成 ISO 15693 模块寻卡功能。主要目的是了解 Microsoft Visual Studio 2010 开发环境及窗体应用程序建立；使程序通过串口与 RFID 实验系统平台建立连接；了解 RFID 实验平台 COM 协议并实现寻卡功能。

本设计所需设备如下。

（1）软件硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），ISO 15693M（高频 13.56MHz）RFID 原理模块（基于 32 位 ARM STM32 嵌入式处理器），ISO 15693 卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计相关原理:上位机应用程序采用 VC#语言开发，遵守 C#编程规范，寻卡功能根据 RFID 原理模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 RFID 原理模块连接；通过串口与 RFID 原理模块进行通信，最终完成寻卡的功能。

2．设计步骤

第一步：创建程序界面。

启动 Microsoft Visual Studio2010 开发平台，选择“文件”→“新建”→“项目”选项，如图 3-8 所示。进入到如图 3-9 所示的界面，选择“Visual C#”→“Windows 窗体应用程序”选项，然后输入名称，再单击“确定”按钮，得到 Windows 窗体，如图 3-10 所示。

图 3-8　新建项目

第二步：设计功能界面。

在“视图”中选择“工具箱”将会出现一些控件，把相应的控件直接拖动到空白的界面中，再根据需要调整它的大小和位置，本实验的功能界面如图 3-11 所示，实现步骤如下。

（1）项目创建完成后，右击界面，在弹出的菜单中选择“属性”选项，将 Text 属性改为“寻卡”。

（2）建立串口连接属性模块：在工具箱中拖出 GroupBox 放在“寻卡”窗体上，并将属性窗口中的 Text 属性修改为“串口操作”；拖出两个 Label，将 Text 属性分别改为“串口号”和“波特率”，拖出两个“ComboBox”，将 Name 属性分别改为“serialPortCombo”和“baudRateCombo”，选中“baudRateCombo”的 ComboBox，设置 Items 的集合为：9600、19200、57600、115200，如图 3-12 所示。

图 3-9　选择 Windows 窗体

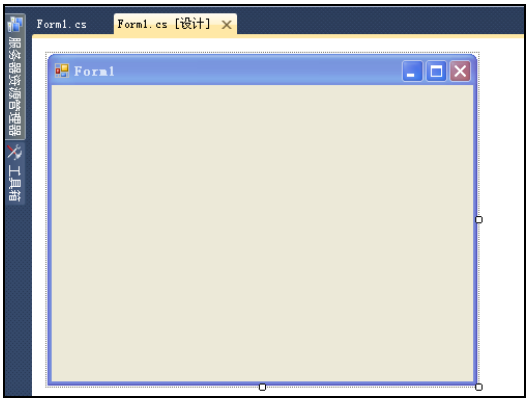


图 3-10　新建窗体

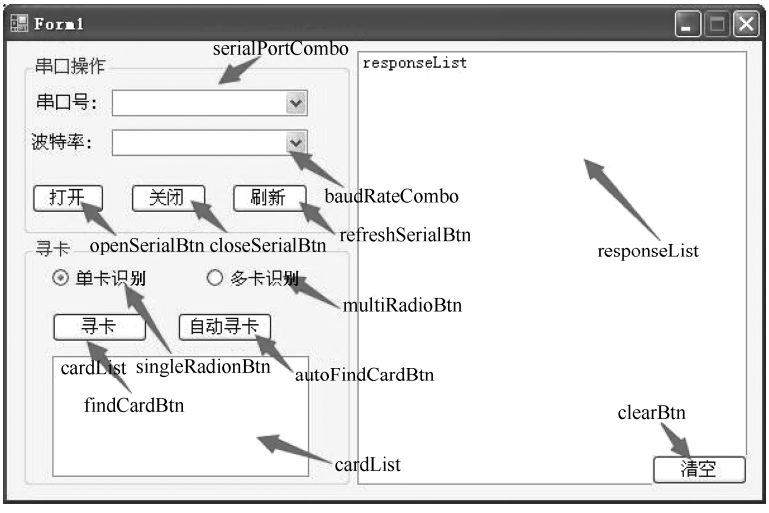


图 3-11　寻卡界面设计

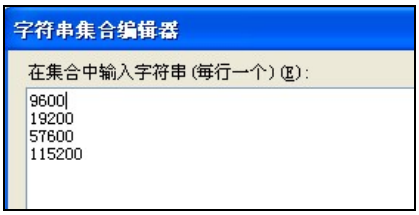


图 3-12　字符串集合编辑器

拖出 3 个 Button，选中相应的控件，修改 Text 属性分别为 “打开”、“关闭”、“刷新”，修改 Name 属性分别为 openSerialBtn、closeSerialBtn、refreshSerialBtn。

（3）建立寻卡功能模块：在工具箱中拖出 GroupBox 控件，放置在窗体上，选中该控件，将 Text 属性改为 “寻卡”；拖出两个 RadioButton，设置 Name 属性分别为 singleRadionBtn 与 multiRadioBtn，设置 Text 分别为 “单卡识别” 与 “多卡识别”；拖出两个 Button 控件，设置 Name 属性分别为 findCardBtn 与 autoFindCardBtn 并修改 Text 属性为 “寻卡” 与 “自动寻卡”；拖出一个 ListBox 控件，设置 Name 属性为 cardList。

（4）建立信息输出模块：在工具箱中拖出 ListBox 控件，放置在窗体上，选中该控件，将 Name 属性改为 responseList；拖出 1 个 Button 控件，设置 Name 属性为 clearBtn，Text 属性为 “清空”。

第三步：编写代码实现功能（此处只附部分主要代码及解析）。

程序执行流程图如图 3-13 所示。

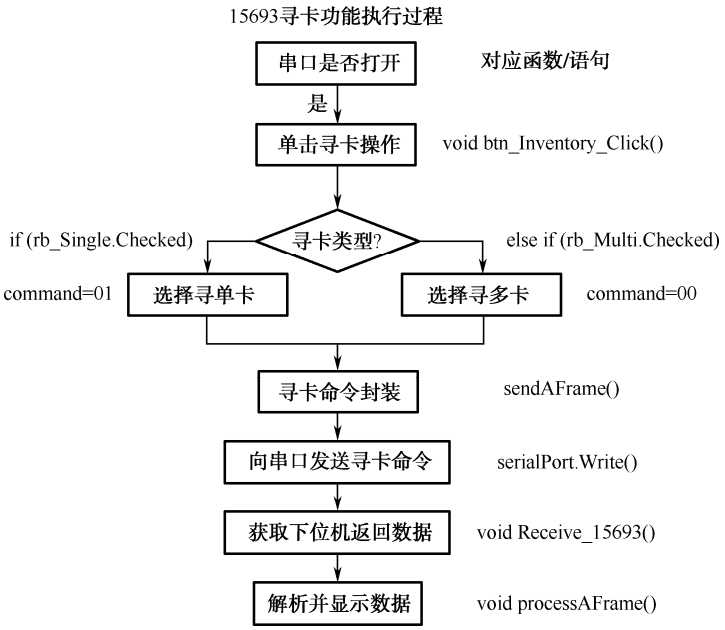


图 3-13　程序执行流程图

部分主要代码解析如下：

```
public void receive_15693(){                    //接收串口返回数据的线程
    String receiveStr = ""; //字符串型数据
    Byte[] receiveData = new Byte[256];         //字节型数组,用于保存串口数据
```

```
        while（isOpen）{   //判断串口是否打开
            try
            {
                while（serialPort.BytesToRead > 0）
                {
                    Int32 count = serialPort.Read（receiveData, 0, receiveData.Length）;
                    //串口发送来的数据保存在 receiveData 中，并返回它的大小
                    for（Int32 i = 0; i < count; i++）
                    {
                        receiveStr += String.Format（"{0:X2}", receiveData[i]）;
                        //把 receiveData 转换成十六进制的字符串
                    }
                    Thread.Sleep（20）;             //当前线程睡眠 20 秒
                }
                if（!receiveStr.Equals（""））
                {
                    Int32 startPos, stopPos;
                    startPos = receiveStr.IndexOf（"EECC"）;       //帧数据开始标志的位置
                    stopPos = receiveStr.IndexOf（"0D0A"）;       //帧数据结束标志的位置
                    while（（startPos >= 0）&&（stopPos > startPos））{
                        String frameStr=receiveStr.Substring（startPos,stopPos-startPos+4）;
                        //截取 EECC 开头，0D0A 结尾的字符串
                        processOneFrame（frameStr）; //处理数据解析
                        ceiveStr=receiveStr.Remove（0, stopPos + 4）;
                        //把 receiveStr 前面的 EECC....0D0A 去掉
                        startPos = receiveStr.IndexOf（"EECC"）; //标识帧头
                        stopPos = receiveStr.IndexOf（"0D0A"）; //标识帧尾
                    }
                }
                receiveStr = "";            //清空保存数据字符串变量
            }
            catch（Exception e）{     //接收数据产生异常,抛出异常,关闭串口
                AddList（String.Format（"串口接收信息异常，异常信息为{0}",e.Message））;
                isOpen = false;
                break;
            }
        }
        try
        {
            serialPort.Close();             //关闭串口
            serialPort = null;              //令串口引用为空
            AddList（String.Format（"串口关闭成功！"））;
        }catch { }
    }
    public void processOneFrame（String frameStr）{         //解析返回来一帧的数据
        Byte[] frameData = new Byte[frameStr.Length/2];     //用两个字符标识一个字节
```

```
    for（int i = 0; i < frameData.Length; i++）
    {
        frameData[i] = Convert.ToByte（frameStr.Substring（i * 2, 2）, 16）;//转换成十六进制数表示
    }
    Byte statusCode = frameData[2];                    //状态码
    Byte hByte = frameData[5];                         //最高有效位
    Byte lByte = frameData[6];                         //最低有效位
    Int32 dataLength = hByte * 256 + lByte;            //数据长度
    String dataByte = frameStr.Substring（14, dataLength * 2）;  //数据
     if（statusCode == 0x10 || statusCode == 0x11 || statusCode == 0x12）
    {
        AddList（String.Format（"命令失败！"））;
    }
    else {
        switch（currentCMD）{          //根据当前操作及命令返回判断命令执行结果
            case 0x01:                 //寻卡返回
            {
                if（statusCode == 0x00）//寻卡成功
                {
                    Byte cardCount =（Byte）（dataLength / 8）;
                    uidList.Clear();               //清空保存 uid 的 List
                    AddList（String.Format（"寻卡执行成功找到了{0}个卡片", cardCount））;
                    for（Byte i = 0; i < cardCount; i++）
                    {
                        String uidStr = "";
                        uidStr = dataByte.Substring（i * 16, 16）;//截取每个 uid 的值
                        uidList.Add（uidStr）;                  //把 uid 保存在 uidList 中
                        AddList（String.Format（uidStr））;  //显示 uid 的值
                    }
                    AddTag();
                }
                else {   //寻卡失败
                    AddList（String.Format（"寻卡执行失败"））;
                }
            }break;
            default: break;
        }
    }
    currentCMD = 0x00;                             //将当前命令更改
}
public void sendOneFrame（Byte command,Byte select,byte []data）{//发送一帧数据方法
    if（!isOpen）{                                 //判断串口是否发开
        AddList（String.Format（"错误：请先通过串口连接设备！"））;
        return;
     }
        Int32 dataLength = data.Length;            //获取发送 Data 区域的数据长度
```

```
            Byte[] frame = new Byte[dataLength + 11];
            frame[0] = 0xEE;                          //第一和第二字节用来标识帧的开始
            frame[1] = 0xCC;
            frame[2] = command;                       //命令
            frame[3] = 0x01;                          //协议
            frame[4] = select;                        //选择
            frame[5] =（Byte）（dataLength / 256）;    //最高有效位
            frame[6] =（Byte）（dataLength % 256）;    //最低有效位
            for（Int32 i = 0; i < dataLength; i++）{
            frame[7 + i] = data[i]; //发送的数据
        }
        frame[7 + dataLength] = 0x00;//倒数第三、第四字节是为了以后扩展协议使用的
        frame[8 + dataLength] = 0x00; //倒数第三、第四字节是为了以后扩展协议使用的
        frame[9 + dataLength] = 0x0D; //倒数第一、第二字节是标识帧结束的
        frame[10 + dataLength] = 0x0A; //标识结束
        String frameStr = "";
        for（Int32 i = 0; i < frame.Length; i++）
        {
             frameStr += String.Format（"{0:X2}", frame[i]）;//转化成十六进制数显示
        }
        try
        {
             serialPort.Write（frame, 0, frame.Length）;      //向串口传输帧数据
             currentCMD = command;                           //将操作命令赋值到当前命令
        }
        catch（InvalidOperationException e）                 //产生异常则抛出异常并关闭串口
        {
            AddList（String.Format（"发送异常{0}",e.Message））;
            isOpen = false;
        }
    }
    private void findCardBtn_Click（object sender, EventArgs e）//寻卡事件
    {
        Byte []data = new Byte[1];                          //一个长度的字节数组
        data[0] = 0x00;                                     //保存 data 数据为 00
        if（singleRadioBtn.Checked）{                       //选择寻单卡
            sendOneFrame（0x01,isSelected,data）;           //发送一帧寻单卡命令
        }
        if（multiRadioBtn.Checked）{                        //选择寻多卡
            sendOneFrame（0x00,isSelected,data）;           //发送一帧寻多卡命令
        }
        currentCMD = 0x01;                                  //标记当前命令为寻卡命令
    }
```

第四步：编译生成程序运行。

编译运行程序，单击“打开”按钮，然后单击“寻卡”按钮，结果如图 3-14 所示。

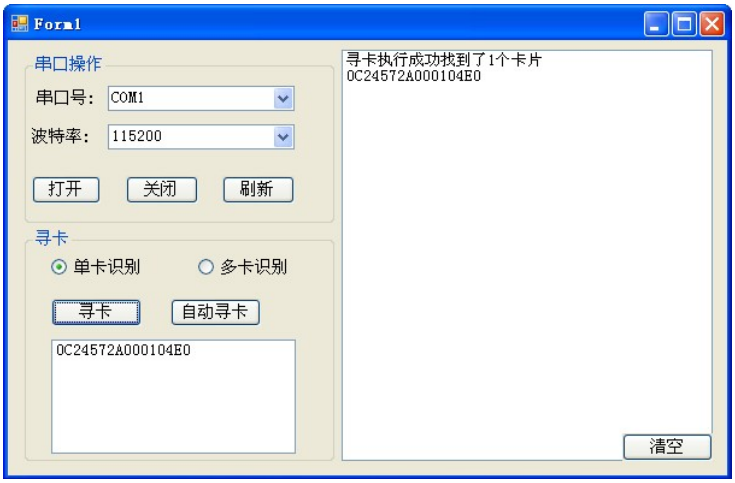


图 3-14　寻卡功能实现

3.2　HF13.56MHz ISO 15693 数据块的读写

3.2.1　协议原理

当附近式耦合设备 VCD（Vicinity Coupling Device）寻址模式标志为“1”时发送一次请求，请求中包含附近式卡 VICC（Vicinity Integrated Circuit Card）的 UID，任何 VICC 在收到寻址标志为“1”的请求后，将收到的 UID 与自身的 UID 作比较，如果匹配，VICC 将执行相应的操作并且向 VCD 发送一个响应。如果不匹配，VICC 不做任何响应。

ISO 15693 标准中规定的命令假定物理内存以固定大小的块（或页）出现。

（1）达到 256 个块可被寻址。

（2）块大小可至 256 位。

（3）这可导致最大的内存容量达到 8KB（64Kb）。

本标准中规定的命令集允许按块操作（读和写）。关于其他操作方式，没有暗示或明示的限制。

每个地址后带 4 个字节存储空间。

3.2.2　操作步骤

（1）打开上位机软件，启动 RFID 原理机的电源，建立通信连接。

（2）读取卡片的 UID。进行数据读取操作，首先需要进行寻卡操作。将卡片放到识别区（如果连接了外接天线，将卡片放置在天线旁边。本书所有实验均采用外接天线为例），打开寻卡命令，选中“单卡识别”单选按钮，单击“寻卡”按钮，寻卡成功返回如图 3-15 所示的信息。

（3）数据块数据读取。选择读取数据模块，选择卡号、填写地址、长度、数量（系统已经默认，可以不改写）。单击“读取多个模块数据”按钮。读取成功返回如图 3-16 所示的信息，地址“0”表示读取第 0 块，数量“2”则是读取两块数据，读取到的数据为 12345678 87654321，表示 0 块的 8 个字节。

图 3-15　寻卡操作

图 3-16　数据块读取

（4）获取卡片的信息，读取到卡的块数和每个块的字节数，如应用族（AFI）、数据存储格式标识 DSFID 等。选择获取系统信息命令，单击“获取系统信息”按钮，读取成功返回如图 3-17 所示的信息，其中“应用族标识”为“00”，“数据存储格式标识”为“00”，“数据块数量”为“1B”个，由于块的编号是从 0～1B 开始的，故共有数据模块 1B+1 个，同理每个数据模块的长度也是 3+1 个，即 4 个字节。

（5）数据块数据写入。单击“写入数据块”按钮，选择卡号，输入地址为 0、块长度为 4、数量 1 块，输入“11223344”，执行“写入多个数据模块”，写入命令执行成功，如图 3-18 所示。

图 3-17　获取系统信息

图 3-18　写入数据块

（6）验证写入是否成功。再次执行步骤（3）读取第 0 块的操作，能够读取到写入的数据则为写入成功。

（7）锁定数据块。选中“锁定数据块”单选按钮，进行锁块操作（锁定数据块后，数据块数据将被锁定，被锁定的数据块只支持读取，不支持写入）。

3.2.3　程序设计：多数据块数据读写功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境上创建 C#窗体应用程序；编写代码，实现多数据

块数据读写功能。主要目的是了解 Microsoft Visual Studio 2010 开发环境及窗体应用程序建立；掌握程序通过串口与 RFID 实验系统平台建立连接并通信的原理和方法；了解 RFID 实验平台 COM 协议并实现多数据块读写功能。

本设计所需设备如下。

（1）硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），ISO 15693M（高频 13.56MHz）RFID 原理模块（基于 32 位 ARM STM32 嵌入式处理器），ISO 15693 卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计主要原理：上位机应用程序采用 C#语言开发，遵守 C#编程规范，读卡功能根据 RFID 原理模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 RFID 原理模块连接；通过串口与 RFID 原理模块进行通信，上位机程序根据 RFID 原理模块 COM 协议实现寻卡功能后，最终完成多数据块读写的功能。

2．设计步骤

第一步：创建 C#窗体应用程序。

启动 Microsoft Visual Studio 2010 开发平台，选择“文件”→“新建”→“项目”命令，如图 3-19 所示。打开如图 3-20 所示的“新建项目”对话框，选择“Visual C#”→“Windows 窗体应用程序”，填写“名称”，单击“确定”按钮，得到 Windows 窗体，如图 3-21 所示。

图 3-19　新建项目

图 3-20　选择 Windows 应用程序

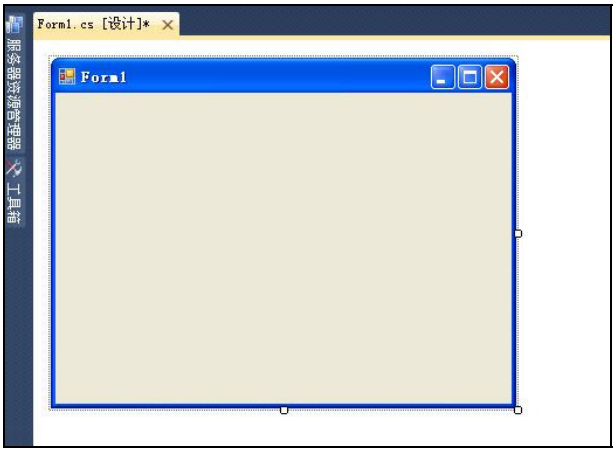


图 3-21　新建窗体

第二步：设计功能界面。

（1）在“视图”中选择“工具箱”命令将会出现一些控件，把相应的控件直接拖动到空白的界面中，再根据需要调整它的大小和位置，本实验的功能界面如图 3-22 所示。

（2）项目创建完成后，右击界面，在弹出的快捷菜单中选择“属性”选项，将属性 Text 改为“多数据块读写”。

（3）建立串口连接属性模块：在工具箱中拖出 GroupBox 放在“多数据块读写”窗体上，并将属性窗口中的 Text 属性修改为“串口操作”；拖出两个 Label，将 Text 属性分别改为“串口号：”和“波特率：”，拖出两个“ComboBox”，将 Name 属性分别改为“serialPortCombo”和“baudRateCombo”，选中“baudRateCombo”的 ComboBox，设置 Items 的集合为 9600、19200、57600、115200，如图 3-23 所示。

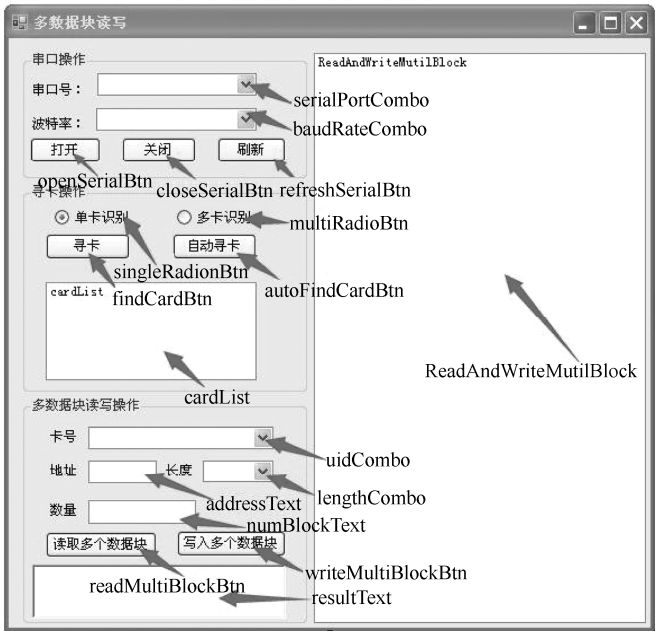


图 3-22　数据块操作界面

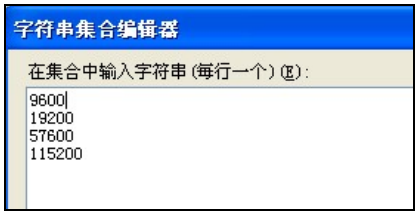


图 3-23　字符串集合编辑器

拖出 3 个 Button，选中相应的控件，修改 Text 属性分别为“打开”、“关闭”、“刷新”，修改 Name 属性分别为“openSerialBtn”、“closeSerialBtn”、“refreshSerialBtn”。

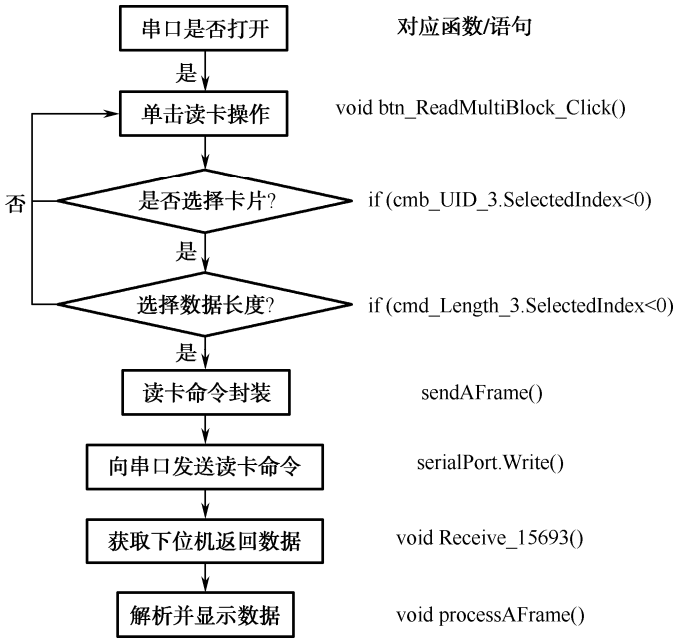
（4）建立寻卡功能模块：在工具箱中拖出 GroupBox 控件，放置在窗体上，选中该控件，将 Text 属性改为“寻卡操作”；拖出两个 RadioButton，设置属性 Name 分别为 singleRadionBtn 与 multiRadioBtn，设置 Text 分别为“单卡识别”与“多卡识别”；拖出两个 Button 控件，设置 Name 属性分别为“findCardBtn”与“autoFindCardBtn”并修改 Text 属性分别为“寻卡”与“自动寻卡”；拖出一个 ListBox 控件，设置 Name 属性为“cardList”。

（5）建立多数据块读写操作模块，在工具箱中拖出 GroupBox 控件，放置在窗体上，选中该控件，将 Text 属性改为“多数据块读写操作”；拖出四个 Label，将 Text 属性分别改为“卡号”、“地址”、“长度”、和“数量”；拖出两个 ComboBox，将 Name 属性分别改为“uidCombo”和“lengthCombo”；拖出两个 TextBox，将 Name 属性分别改为“addressText”和“numBlockText”；拖出两个 Button 控件，设置 Name 属性分别是“readMultiBlockBtn”和“writeMultiBlockBtn”，并设置 Text 属性分别为 “读取多个数据块”和“写入多个数据块”；拖出一个 RichTextBox 空间，设置属性为“resultText”。

（6）建立信息输出模块：在工具箱中拖出 ListBox 控件，放置在窗体上，选中该控件，将 name 属性改为“ReadAndWriteMultiBlock”；拖出 1 个 Button 控件，设置 Name 属性为“clearBtn”，Text 属性为 “清空”。

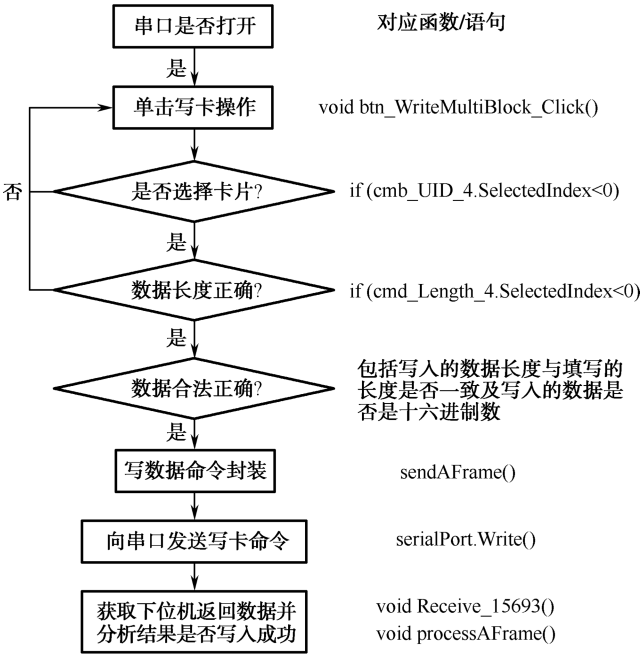
第三步：编写代码实现功能（此处只附部分主要代码及解析）。

程序执行流程图如图 3-24 所示。



（a）15693读卡功能执行过程

图 3-24　程序执行流程图



（b）15693写卡功能执行过程

图 3-24　程序执行流程图（续）

部分主要代码解析如下：

```
// 读取多个卡按钮监听事件
private void btn_ReadMultiBlock_Click（object sender, EventArgs e）
{
    if（uidList.Count < 1）                    //判断是否寻到卡片
    {
        AddList（"错误：未发现卡片，请先寻卡！"）;
        return; //返回
    }
    if（cmb_UID_3.SelectedIndex < 0）          //判断是否选择操作的卡片
    {
        AddList（"错误：请选择您要读取的卡片！"）;
        cmb_UID_3.Focus();
        return;
    }
    if（cmd_Length_3.SelectedIndex < 0）       //判断是否选择了正确的数据长度
    {
        AddList（"错误：请选择正确的数据块长度！"）;
        cmd_Length_3.Focus();
        return;
    }
    String struid = cmb_UID_3.Text.Trim();     //字符串类型,保存卡片
    Byte command = 0x07;                       //读卡命令
    Byte[] data = new Byte[10];                //字节数组,10 字节长度
    for（Byte i = 0; i < 8; i++）              //其中 8 个字节保存卡号
    {
```

```
            data[i] = Convert.ToByte（struid.Substring（i * 2, 2）, 16）；
        }
        data[8] = Byte.Parse（txt_Address_3.Text.Trim()）；      //第 9 个字节保存地址
        data[9] = Byte.Parse（txt_Number_3.Text.Trim()）；       //第 10 个字节保存数量
        sendAFrame（command, isSelected, data）；                //发送读卡命令
    }
    //写入数据按钮事件
    private void button2_Click（object sender, EventArgs e）
    {
        if（uidList.Count < 1）{                                 //判断是否寻卡成功
            AddList（String.Format（"错误：还没有卡片，请先寻卡！"））；
            return;
        }
        if（uidCombo.SelectedIndex < 0）                         //判断是否选择卡片
        {
            AddList（String.Format（"错误：你还没有选择卡片，请先选择！"））；
            uidCombo.Focus();
            return;
        }
        if（lengthCombo.SelectedIndex < 0）                      //判断数据长度是否合法
        {
            AddList（String.Format（"请选择正确的数据块长度！"））；
            lengthCombo.Focus();
            return;                          //非法则跳出
        }
        Byte address;                        //起始地址
        Byte numBlock;                       //数据块数量
        Byte lengthBlock;                    //数据块长度
        try
        {
            address = Byte.Parse（addressText.Text.Trim()）；    //保存写入的地址
        }
        catch                                //在保存地址过程中出现异常则抛出异常并返回
        {
            AddList（String.Format（"错误:地址输入格式出错！"））；
            addressText.Focus();
            return;
        }
        try
        {
            numBlock = Byte.Parse（numBlockText.Text.Trim()）；  //保存写入的数量
        }
        catch                                //在保存数量过程中出现异常则抛出异常并返回
        {
            AddList（String.Format（"错误:数量格式出错！"））；
            numBlockText.Focus();
```

```
        return;
    }
    try
    {
        lengthBlock = Byte.Parse（lengthCombo.Text.Trim()）;
    }
    catch
    {
        AddList（String.Format（"错误:数量格式出错！"））;
        lengthCombo.Focus();
        return;
    }
    Byte numByte =（Byte）（numBlock * lengthBlock）; //数据块总字节数
    Byte[] dataForWrite = new Byte[numByte];    //新建字节数组,长度为写入数据的长度
    String strForWrite = resultText.Text.Trim();    //写入的数据
    try   //将写入的数据进行数据类型转换
    {
        for（int i = 0; i < dataForWrite.Length; i++）
        {
            dataForWrite[i] = Convert.ToByte（strForWrite.Substring（2 * i, 2）, 16）;
        }
    }
    catch（System.ArgumentOutOfRangeException w）//若转换失败,抛出异常
    {    AddList（String.Format（"错误：写入的数据长度应该为{0}*{1}={2}Byte",
                    numBlock, lengthBlock, numByte））;
        resultText.SelectAll();
        resultText.Focus();
        return;
    }
    catch（System.Exception er）
    {    AddList（String.Format（"错误：输入的数据应该是{0}*{1}={2}的十六进制数",
                    numBlock, lengthBlock, numByte））;
        resultText.SelectAll();
        resultText.Focus();
        return;
    }
    String uidStr = uidCombo.Text.Trim();   //写入的卡号
    Byte command = 0x08;                            //写卡命令
    Byte []data = new Byte[10+numByte];             //data 区域
    for（int i = 0; i < 8; i++）{
        data[i] = Convert.ToByte（uidStr.Substring（2*i,2）,16）;
    }
    data[8] = address;       //地址
    data[9] = numBlock;    //块数
    for（int i = 0; i < dataForWrite.Length; i++）
    {       //把要传送的数据发到一帧中
```

```
                data[10 + i] = dataForWrite[i];
            }
            sendOneFrame（command, isSelected, data）; //发送写卡命令
        }
```

第四步：编译生成程序运行。

在运行的程序窗口中，选择正确的串口号和波特率，执行“打开”→“寻卡”→“读取多个数据块”→“写入多个数据块”命令，然后输入想写入的数据，再单击“写入多个数据块”按钮，最后再把写入的数据读出来，如图 3-25 所示。

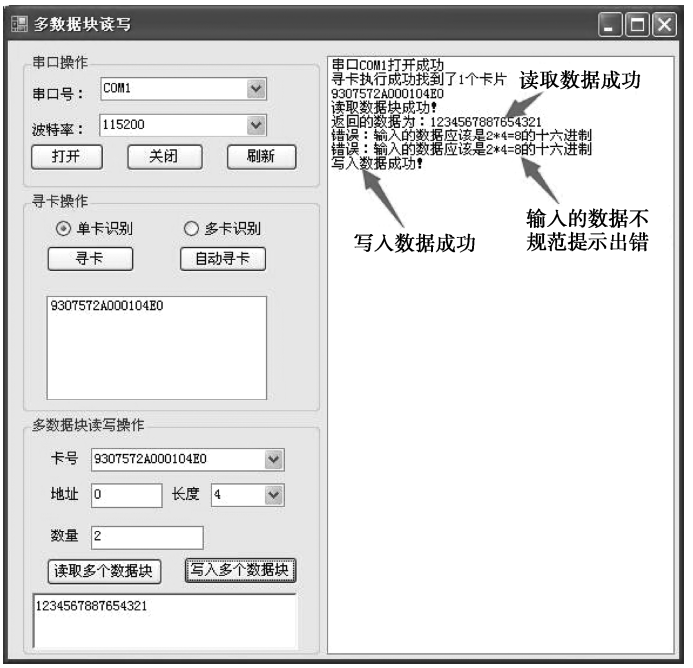


图 3-25　数据块操作功能实现

3.3　HF13.56MHz ISO 15693 应用族标识

3.3.1　协议原理

15693 协议电子标签数量众多，应用范围广泛。为了区分不同行业中的电子标签，ISO 用一个字节的 AFI（Application Family Identifier）来区分不同行业中的电子标签。AFI 的高半字节表示主要行业，低半字节表示主要行业中的细分行业。其中 AFI=00H 表示所有行业。需要注意的是并不强制要求电子标签支持 AFI，电子标签是否支持 AFI 是可选的，在收到 Inventory 清点命令后，如果标签不支持 AFI，则标签必须立刻做出应答；如果支持 AFI，则只有收到的 AFI 与标签存储的 AFI 一致才做出应答。

应用族标识符 AFI（Application Family Identify）代表由 VCD 锁定的应用类型，VICC 只有满足所需的应用准则才能从出现的 VICC 中被挑选出。

AFI 被相应的命令锁定。

AFI 被编码在一个字节里，由两个半字节组成。

AFI 的高位半字节用于编码一个特定的或所有的应用族，定义见表 3-3。

表 3-3　VICC 的响应方式

AFI 高半字节	AFI 低半字节	VICC 的响应方式	举例/注释
0	0	所有族和子族	无可用预选
x	0	x 族的子族	无可用预选
x	y	x 族的仅第 y 个子族	
0	y	仅子族 y 所有权	
1	0，y	运输	批量运输，公交，航空
2	0，y	金融	IEP，银行，零售
3	0，y	标识	进入控制
4	0，y	无线电通信	公用电话，GSM
5	0，y	医疗	
6	0，y	多媒体能	
7	0，y	游戏	
8	0，y	数据存储	
9	0，y	条款管理	
A	0，y	快递包裹	
B	0，y	邮政服务	
C	0，y	航空运输	
D	0，y	RFU	
E	0，y	RFU	
F	0，y	RFU	

注：x='1'～'F'，y='1'～'F'。

AFI 的低位半字节用于编码一个特定的或所有的应用子族。子族不同于 0 的编码有其自己的所有权。

VICC 支持的 AFI 是可选的。

假如 VICC 不支持 AFI，并且假如 AFI 标识已设置，VICC 将不应答任何请求中的 AFI 值。

假如 VICC 支持 AFI，VICC 将根据表中匹配规则作出应答。

3.3.2　操作步骤

（1）打开上位机软件，启动 RFID 原理机的电源，建立通信连接。

（2）读取卡片的 UID。进行数据读取操作，首先需要进行寻卡操作。将卡片放到识别区（如果连接了外接天线，将卡片放置在天线旁边。本书所有实验均采用外接天线为例），打开寻卡命令，选中“单卡识别”单选按钮，单击“寻卡”按钮执行寻卡操作，寻卡成功返回如图 3-26 所示的信息。

图 3-26　寻卡操作

（3）查询当前应用族标识。选择获取系统信息，选择卡号，单击“获取系统信息”按钮进行信息获取。如图 3-27 所示，获取到的应用族标识为“00”。

图 3-27　获取系统信息

（4）写入应用族标识。如图 3-28 所示，在功能界面上选择“写入应用族标识”。弹出写入应用族标识功能执行区，选择进行操作的标签号并输入写入的应用族标识（如写入“01”），

单击“写入应用族标识”按钮，命令执行成功。

图 3-28　写入应用族标识

（5）查询卡片信息，验证其是否写入成功。查询到应用族标识为“01”，证明应用族标识写入成功，如图 3-29 所示。

图 3-29　查询写入的应用族标识

（6）应用族标识锁定。如图 3-30 所示，在功能界面上选中“锁定应用族标识”单选按钮。弹出锁定应用族标识功能执行区，选择进行操作的标签号，单击“锁定应用族标识”按钮，命令执行成功（操作时慎用此功能）。锁定应用族标识后，被锁定的卡片不再支持写入应用族标识操作。

图 3-30　锁定应用族标识

3.3.3　程序设计：应用族标识写入及锁定功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境下创建 C#窗体应用程序；编写代码，实现应用族标识写入及锁定功能。主要目的是了解 Microsoft Visual Studio 2010 开发环境及窗体应用程序建立；掌握程序通过串口与 RFID 实验系统平台建立连接并通信的原理和方法；了解 RFID 实验平台 COM 协议并实现应用族标识写入及锁定功能。

本设计所需设备如下。

（1）硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），ISO 15693M（高频 13.56MHz）RFID 原理模块（基于 32 位 ARM STM32 嵌入式处理器），ISO 15693 卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计主要原理：上位机应用程序采用 C#语言开发，遵守 C#编程规范，卡操作功能根据 RFID 原理模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 RFID 原理模块连接；上位机程序根据 RFID 原理模块 COM 协议实现寻卡功能后，最终完成应用族标识写入及锁定的功能。

2．设计步骤

第一步：创建 C#窗体应用程序。

启动 Microsoft Visual Studio 2010 开发平台，创建一个如图 3-31 所示的 C#窗体应用程序，命名为“ISO 15693_WriteAFI”。

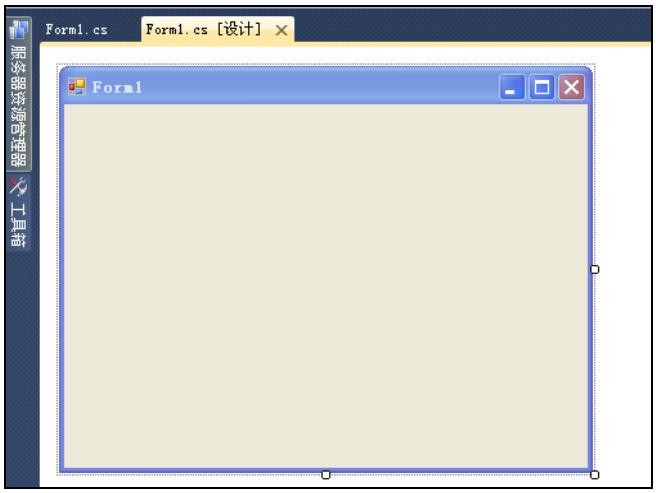


图 3-31　C#窗体应用程序

第二步：设计功能界面。

（1）选中新建的窗体，右击界面，在弹出的快捷菜单中选择“属性”选项，将属性 Text 改为“应用族标识操作”。

（2）建立串口连接属性模块。在工具箱中拖出 GroupBox 放在“应用族标识操作”窗体上，并将属性窗口中的 Text 属性修改为“串口操作”；拖出两个 Label，将 Text 属性分别改为“串口号：”和“波特率：”，拖出两个“ComboBox”，将 Name 属性分别改为“serialPortCombo”和“baudRateCombo”，选中“baudRateCombo”的 ComboBox，设置 Items 的集合为 9600、19200、57600、115200，如图 3-32 所示。

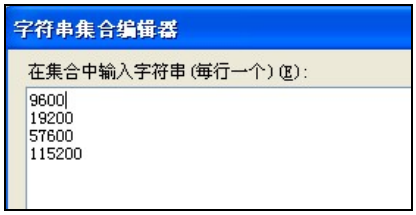


图 3-32　字符串集合编辑器

拖出 3 个 Button，选中相应的控件，修改 Text 属性分别为“打开”、“关闭”、“刷新”，修改 Name 属性分别为“openSerialBtn”、“closeSerialBtn”、“refreshSerialBtn”。

（3）建立寻卡功能模块。在工具箱中拖出 GroupBox 控件，放置在窗体上，选中该控件，将 Text 属性改为“寻卡操作”；拖出两个 RadioButton，设置 Name 属性分别为“singleRadionBtn”与“multiRadioBtn”，设置 Text 分别为“单卡识别”与“多卡识别”；拖出两个 Button 控件，设置 Name 属性分别为“findCardBtn”与“autoFindCardBtn”并修改 Text 属性分别为“寻卡”与“自动寻卡”；拖出一个 ListBox 控件，设置 Name 属性为“cardList”。

（4）建立应用族功能操作模块。在工具箱中拖出 GroupBox 控件，放置在窗体上，选中该控件，将 Text 属性改为“应用族标识操作”； 拖出三个 Label，将 Text 属性分别改为“卡号:”、“应用族标识:0x”和“Hex”；拖入一个“ComboBox”，将属性 Name 分别改为“cmb_UID_9”；拖入一个 TextBox，将 Name 属性设为“txt_AFI”，将 Text 属性设为“00”；拖入两个 Button，

将 Name 属性分别设为“btn_WriteAFI”和“btn_LockAFI”，将 Text 属性分别设为“写入应用族标识”和“锁定应用族标识”。

（5）建立信息输出模块：在工具箱中拖出 ListBox 控件，放置在窗体上，选中该控件，将 Name 属性改为“WriteMAFIlst”；拖出 1 个 Button 控件，设置 Name 属性为 clearBtn，Text 属性为“清空”。

完成上述步骤后，得到的界面如图 3-33 所示。

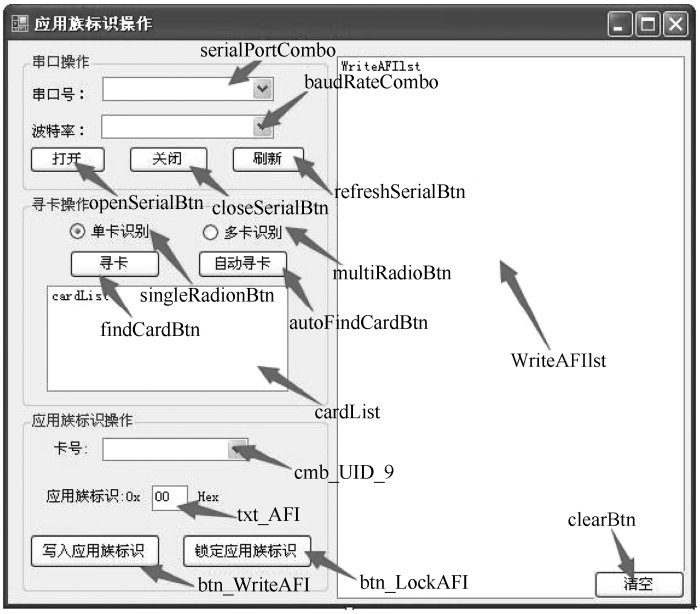


图 3-33　应用族标识操作界面设计

第三步：编写代码实现功能（此处只附部分主要代码及解析）。

15693 应用族标识操作功能执行过程如图 3-34 所示。

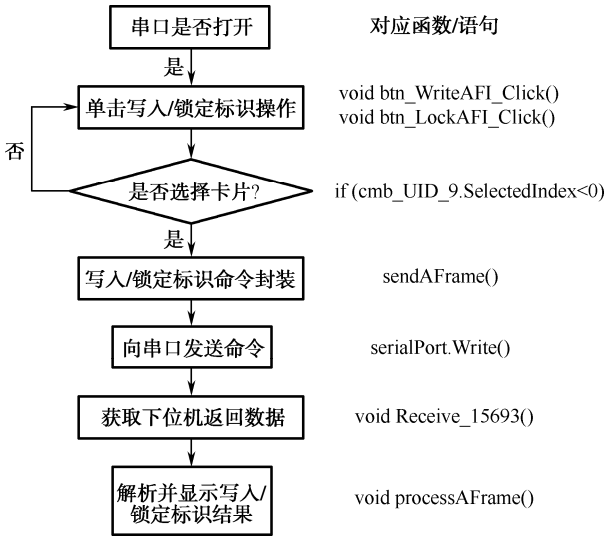


图 3-34　15693 应用族标识操作功能执行过程

部分主要代码解析如下：

```
// 写入应用族标识符监听
private void btn_WriteAFI_Click（object sender, EventArgs e）
{
    if（cmb_UID_9.SelectedIndex < 0）      //判断是否选择卡片
    {
        AddList（"错误：请选择您要操作的卡片！"）;
        cmb_UID_9.Focus();
        return; //若没有选择卡片,则跳出事件,不再往下执行
    }
    String struid = cmb_UID_9.Text.Trim();        //字符串,保存卡号
    Byte command = 0x05;              //标识写入应用族标识命令
    Byte[] data = new Byte[9];         //字节数组,保存卡号和写入的 AFI
    for（Byte i = 0; i < 8; i++）          //卡号长度 8 个字节
    {
        data[i] = Convert.ToByte（struid.Substring（i * 2, 2）, 16）;   //保存卡号
    }
    data[8] = Convert.ToByte（txt_AFI.Text.Trim(), 16）;   //保存应用族标识
    sendOneFrame（command, isSelected, data）;//发送一帧写入应用族标识命令
}
// 锁定应用族标识监听
private void btn_LockAFI_Click（object sender, EventArgs e）
{
    if（cmb_UID_9.SelectedIndex < 0）      //判断是否选择了卡号
    {
        AddList（"错误：请选择您要操作的卡片！"）;   //显示错误信息
        cmb_UID_9.Focus();
        return;// 若没有选择卡片,则跳出事件,不再往下执行
    }
    if（MessageBox.Show（"AFI 锁定是不可逆的，您确认要锁定吗？", "提示",
            MessageBoxButtons.YesNo）== DialogResult.No）//弹出窗口提示信息
        return;//若单击了‘否’，则返回，不再往下执行锁定 AFI
    String struid = cmb_UID_9.Text.Trim();                    //字符串,保存卡号
    Byte command = 0x06;                                      //命令,锁定命令
    Byte[] data = new Byte[8];                                //字节数组,用于保存卡号
    for（Byte i = 0; i < 8; i++）                              //卡号长度为 8 个字节
    {
        data[i] = Convert.ToByte（struid.Substring（i * 2, 2）, 16）; //保存卡号
    }
    sendOneFrame（command,isSelected,data）;//发送锁定应用族标识命令,应用族标识将被锁定
}
```

第四步：编译生成程序运行。

在运行的程序窗口中，选择正确的串口号和波特率。单击“寻卡”按钮进行寻卡操作。寻卡成功后，进行写入标识操作。写入成功返回如图 3-35 所示的信息。

图 3-35　应用族功能操作实现

3.4　HF13.56MHz ISO 15693 存储格式标识

3.4.1　协议原理

15693 国际标准规定了一个字节的可选的数据存储格式识别符（DSFID），用来区分标签中不同的数据存储格式。如果标签支持 DSFID，在清点命令中标签将返回一个非零的 DSFID，读写器可根据此判断射频场中的标签是否具有期望的数据格式。数据存储格式识别符（DSFID）特点如下。

数据存储格式标识符指出了数据在 VICC 内存中是怎样构成的。

DSFID 被相应的命令编程和锁定，DSFID 被编码在一个字节中。DSFID 允许及时知道数据的逻辑组织。

假如 VICC 不支持 DSFID 的编程,VICC 将以“0”值作为应答。

3.4.2　操作步骤

（1）打开上位机软件，启动 RFID 原理机的电源，建立通信连接。

（2）读取卡片的 UID。参照前面的操作步骤进行标签的 UID 读取操作。

（3）查询当前数据存储格式标识。选择获取系统信息，选择卡号，再单击“获取系统信息”按钮进行信息获取。如图 3-36 所示，获取到的数据存储格式标识为“00”。

（4）写入数据存储格式标识。选择“写入数据存储格式标识”，选择写入的卡号，输入要写入的标识（任意一个字节的十六进制数，这里以写入的标识“02”为例），再单击“写入数据存储格式标识”按钮，如图 3-37 所示。

图 3-36　获取系统中存储格式标识信息

图 3-37　写入数据存储格式标识成功

（5）锁定数据存储格式标识。选择“锁定数据存储格式标识”，选择锁定的标签号，进行锁块操作（此功能操作者慎用），锁定存储格式标识后，存储格式标识将被锁定，被锁定的存储格式标识只支持读取，不支持写入，如图 3-38 所示。

图 3-38　锁定存储格式标识

3.4.3　程序设计：存储格式标识写入及锁定功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境下创建 C#窗体应用程序；编写代码，实现存储格式标识写入及锁定功能。主要目的是了解 Microsoft Visual Studio 2010 开发环境及窗体应用程序建立；掌握程序通过串口与 RFID 实验系统平台建立连接并通信的原理和方法；了解 RFID 实验平台 COM 协议并实现存储格式标识写入及锁定功能。

本设计所需设备如下。

（1）硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），ISO 15693M（高频 13.56MHz）RFID 原理模块（基于 32 位 ARM STM32 嵌入式处理器），ISO 15693 卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计主要原理：上位机应用程序采用 C#语言开发，遵守 C#编程规范，卡操作功能根据 RFID 原理模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 RFID 原理模块连接；上位机程序根据 RFID 原理模块 COM 协议实现寻卡功能后，通过串口与 RFID 原理模块进行通信，最终完成存储格式标识写入及锁定的功能。

2．设计步骤

第一步：创建 C#窗体应用程序。

启动 Microsoft Visual Studio2010 开发平台，创建一个如图 3-39 所示的 C#窗体应用程序，命名为“ISO 15693_WriteDSFID”。

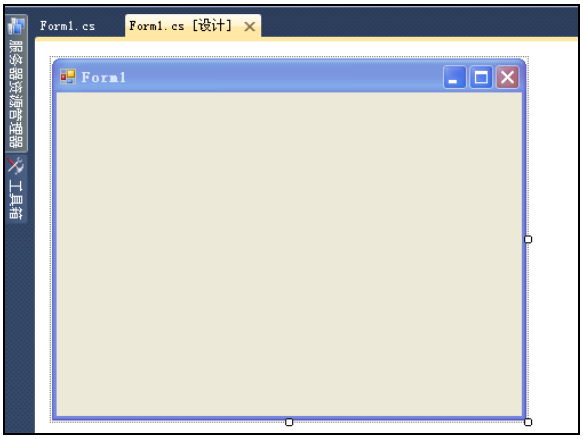


图 3-39　新建窗体应用程序

第二步：设计功能界面。

（1）选中新建的窗体，右击界面，在弹出的快捷菜单中选择“属性”选项，将属性 Text 改为“数据存储格式标识操作”。

（2）建立串口连接属性模块。在工具箱中拖出 GroupBox 放在“数据存储格式操作”窗体上，并将属性窗口中的 Text 属性修改为“串口操作”；拖出两个 Label，将 Text 属性分别改为“串口号：”和“波特率：”，拖出两个“ComboBox”，将 Name 属性分别改为“serialPortCombo”和“baudRateCombo”，选中“baudRateCombo”的 ComboBox，设置 Items 的集合为 9600、19200、57600、115200，如图 3-40 所示。

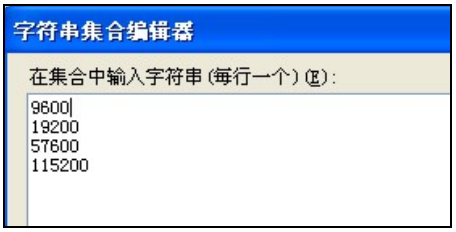


图 3-40　字符串集合编辑器

拖出 3 个 Button，选中相应的控件，修改 Text 属性分别为“打开”、“关闭”、“刷新”，修改 Name 属性分别为“openSerialBtn”、“closeSerialBtn”、“refreshSerialBtn”。

（3）建立寻卡功能模块。在工具箱中拖出 GroupBox 控件，放置在窗体上，选中该控件，将 Text 属性改为“寻卡操作”；拖出两个 RadioButton，设置属性 name 分别为“singleRadionBtn”与“multiRadioBtn”，设置 Text 属性分别为 “单卡识别”与“多卡识别”；拖出两个 Button 控件，设置 Name 属性分别“findCardBtn”与“autoFindCardBtn”并修改 Text 属性分别为“寻卡”与“自动寻卡”；拖出一个 ListBox 控件，设置 Name 属性为“cardList”。

（4）建立数据存储格式标识操作模块。工具箱中拖出 GroupBox 控件，放置在窗体上，选中该控件，将 Text 属性改为“存储格式标识操作”；在 GroupBox 控件中拖入 3 个 Label，将属性分别设为“卡号:”、“ 数据存储格式标识:”和“Hex”；拖入一个“ComboBox”，将 Name 属性设为“cmb_UID_11”；拖入一个“TextBox”，将 Name 属性设为“txt_DSFID”，Text 属性设为“00”；拖入两个 Button，将 Name 属性分别设为“btn_WriteDSFID”和“btn_LockDSFID”，将 Text 属性分别设为“写入数据存储格式标识”和“锁定数据存储格式标识”。

（5）建立信息输出模块：在工具箱中拖出 ListBox 控件，放置在窗体上，选中该控件，将 Name 的属性改为“WriteDSFIDlst”；拖出 1 个 Button 控件，设置 Name 属性为 clearBtn，Text 属性为“清空”。

完成上述步骤后，得到的界面如图 3-41 所示。

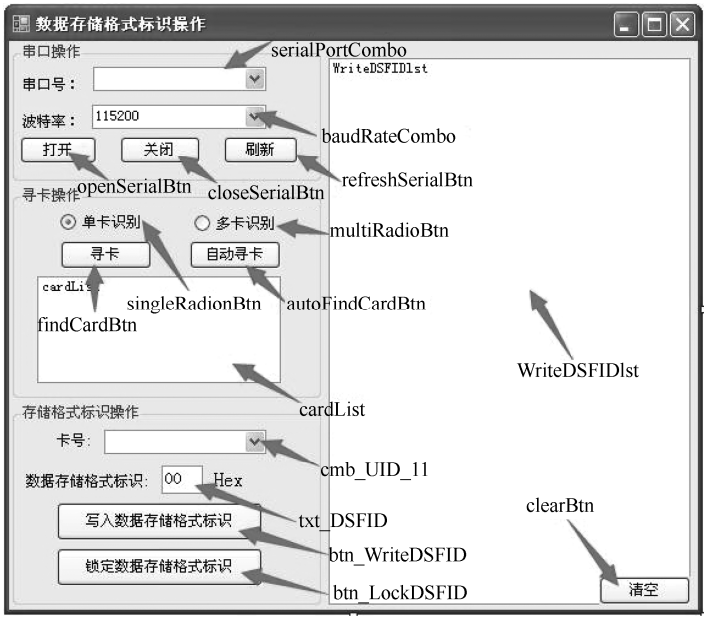


图 3-41　存储格式标识操作界面

第三步：编写代码实现功能（此处只附部分主要代码及解析）。

15693 存储格式标识功能操作执行过程如图 3-42 所示。

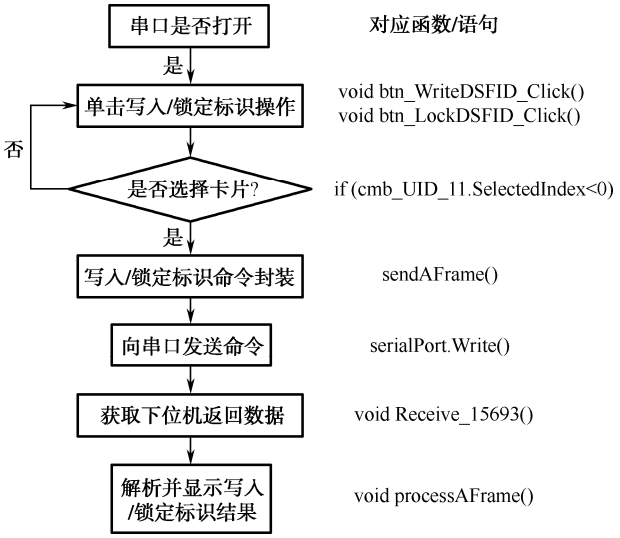


图 3-42　15693 存储格式标识功能操作执行过程

部分主要代码解析如下：

```
//写入数据存储格式标识
private void btn_WriteDSFID_Click（object sender, EventArgs e）
{
    if（cmb_UID_11.SelectedIndex < 0）    //判断是否选择了卡片
```

```
        {
            AddList（"错误：请选择您要操作的卡片！"）;
            cmb_UID_11.Focus();
            return;                 //返回
        }
        String struid = cmb_UID_11.Text.Trim();       //保存卡片
        Byte command = 0x0A;                          //写入标识命令
        Byte[] data = new Byte[9];                    //字节数组,长度为 9 字节
        for（Byte i = 0; i < 8; i++）                 //数组前 8 字节保存卡号
        {
            data[i] = Convert.ToByte（struid.Substring（i * 2, 2）, 16）;
        }
        data[8] = Convert.ToByte（txt_DSFID.Text.Trim(), 16）;      //数组第 9 字节保存写入的标识
        sendOneFrame（command, isSelected, data）;                  //发送一帧写入标识命令
    }

    // 锁定数据存储格式标识按钮监听
    private void btn_LockDSFID_Click（object sender, EventArgs e）
    {
        if（cmb_UID_11.SelectedIndex < 0）//判断是否选择卡片
        {
            AddList（"错误：请选择您要操作的卡片！"）;
            cmb_UID_12.Focus();
            return;
        }
        if（MessageBox.Show（"DSFID 锁定是不可逆的，您确认要锁定吗？", "提示",
                        MessageBoxButtons.YesNo）== DialogResult.No）
            return;
        String struid = cmb_UID_11.Text.Trim();       //保存卡号
        Byte command = 0x0B;                          //锁定标识命令
        Byte[] data = new Byte[8];                    //保存卡号
        for（Byte i = 0; i < 8; i++）
        {
            data[i] = Convert.ToByte（struid.Substring（i * 2, 2）, 16）;   //数据类型转换并保存
        }
        sendOneFrame（command, isSelected, data）;                  //发送锁定标识命令
    }
```

第四步：编译生成程序运行。

在运行的程序窗口中，选择正确的串口号和波特率，单击“打开”按钮打开串口，单击“寻卡”按钮进行寻卡操作。然后进行数据存储格式标识写入标识操作。执行结果如图 3-43 所示。

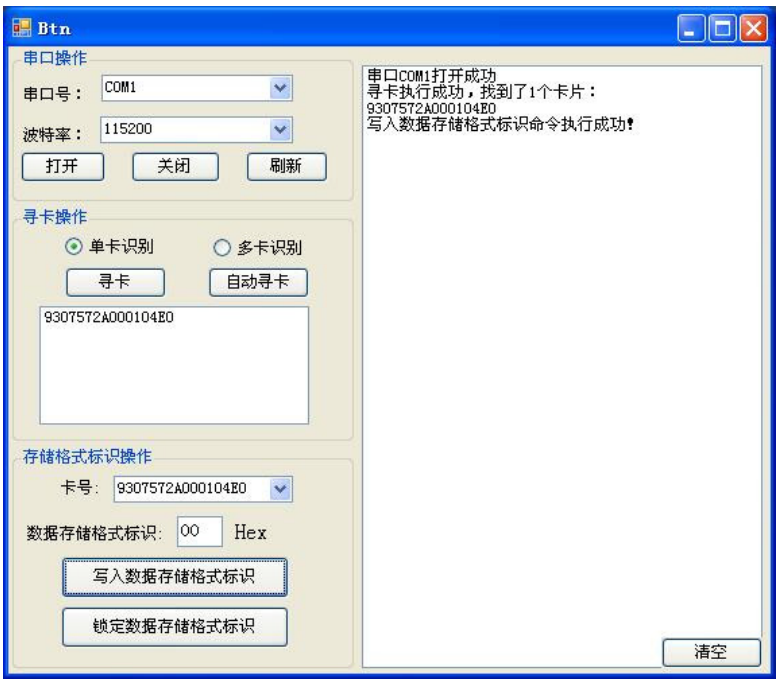


图 3-43　数据存储格式标识操作实现

3.5　HF13.56MHz ISO 15693 防冲撞

3.5.1　协议原理

1．内容

通过使用带 Anticollision 处理过程的指令和不带 Anticollision 处理过程的指令读取多个标签，以及使用不带 Anticollision 处理过程的指令读取多个标签，比较其指令和读取的结果。分析实验数据，总结防冲撞机理，掌握指令产生的作用。

2．原理

当读写器模块 VCD 同时读取到多个卡片 VICC 时，需要利用读写器与标签之间的无线信道进行数据交换，并区分多个标签之间的数据来源与归属，当数据不能按照各个卡片的识别标志进行读写时，就会发生标签碰撞现象。防止 RFID 碰撞的技术是解决标签读写信道的仲裁机制。

在 slot 为 16 的情况下，在一次典型的防冲突序列中，有可能发生的情况如下。

（1）VCD 发送一次目录请求，在一帧内，由 EOF 结束。slot 的数量是 16。

（2）VICC 1 在 slot 0 发送其响应。它是唯一发送响应的 VICC，因此不会发生冲突，VCD 收到它的 UID 并为其注册。

（3）VCD 发送一个 EOF，意思是接通到下一个 slot。

（4）在 slot 1，两个 VICC 2 和 3 传输它们的响应，产生一次冲突。

（5）VCD 发送一个 EOF，意思是接通至下一个 slot。

（6）在 slot 2，没有 VICC 传输响应。因此 VCD 检测不到 VICC SOF，而是通过发送一个

EOF 接通至下一个 slot。

（7）在 slot 3，来自 VICC 4 和 5 的响应会引起另一次冲突。

（8）VCD 决定发送一个寻址请求（如一个读块请求）给 VICC 1，其 UID 已被正确接收。

（9）所有的 VICC 检测到 SOF，将退出防冲突序列。它们处理这个请求，因为请求地址是配给 VICC 1 的，只有 VICC 1 可传输其响应。

（10）所有的 VICC 准备接收下一个请求。假如它是一个目录命令，slot 编写序列号的方式是从 0 开始。

3.5.2 操作步骤

（1）打开上位机软件，启动 RFID 电源，建立通信连接。

（2）将两张 ISO 15693 卡片放置在 RFID 原理机天线旁，选中“测试防冲突”单选按钮，单击“发送使用防冲突算法的寻卡命令”按钮，可以成功寻到两张卡片，如图 3-44 所示。

图 3-44　测试使用防冲突算法寻卡

（3）单击“发送不使用防冲突算法的寻卡命令”按钮，可以发现两张卡片的信息都无法读取到，如图 3-45 所示。

（4）取走其中一张卡，单击“发生不使用防冲突算法的寻卡命令”按钮，可以读取到卡片的信息，如图 3-46 所示。

图 3-45　测试不使用防冲突算法寻卡（1）

图 3-46　测试不使用防冲突算法寻单卡（2）

3.5.3　程序设计：防冲突测试功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境下创建 C#窗体应用程序；编写代码，实现防冲突

测试功能。主要目的是了解 Microsoft Visual Studio 2010 开发环境及窗体应用程序建立；掌握程序通过串口与 RFID 实验系统平台建立连接并通信的原理和方法；了解 RFID 实验平台 COM 协议并实现防冲突测试功能。

本设计所需设备如下。

（1）硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），ISO 15693M（高频 13.56MHz）RFID 原理模块（基于 32 位 ARM STM32 嵌入式处理器），ISO 15693 卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计主要原理：上位机应用程序采用 C#语言开发，遵守 C#编程规范，卡操作功能根据 RFID 原理模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 RFID 原理模块连接；上位机程序根据 RFID 原理模块 COM 协议实现寻卡功能后，最终完成防冲突测试功能。

2．设计步骤

第一步：创建 C#窗体应用程序。

启动 Microsoft Visual Studio 2010 开发平台，创建一个如图 3-47 所示的 C#窗体应用程序，命名为“ISO 15693_TestAnticoll”。

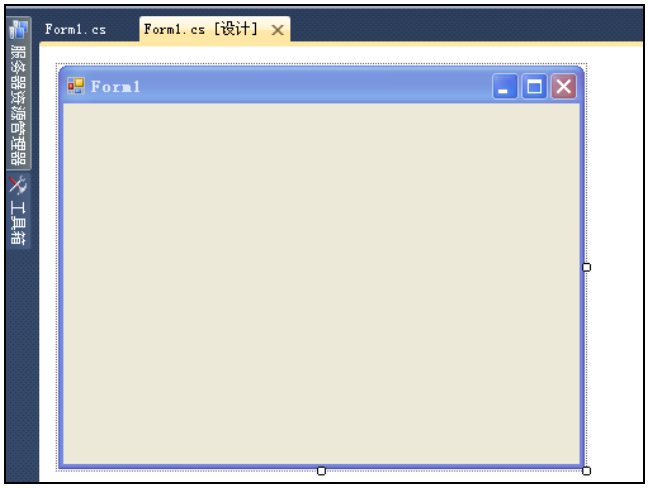


图 3-47　新建窗体应用程序

第二步：设计功能界面。

（1）选中新建的窗体，右击界面，在弹出的快捷菜单中选择“属性”选项，将属性 Text 改为“测试防冲突”。

（2）建立串口连接属性模块。在工具箱中拖出 GroupBox 放在“数据存储格式操作”窗体上，并将属性窗口中的 Text 属性修改为“串口操作”；拖出两个 Label，将 Text 属性分别改为“串口号：”和“波特率：”，拖出两个“ComboBox”，将 Name 属性分别改为“serialPortCombo”和“baudRateCombo”，选中“baudRateCombo”的 ComboBox，设置 Items 的集合为 9600、19200、57600、115200，如图 3-48 所示。

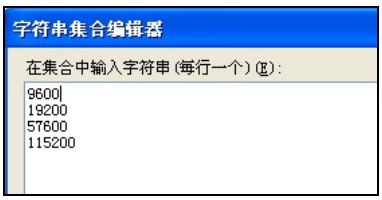


图 3-48　字符串集合编辑器

拖出 3 个 Button，选中相应的控件，修改 Text 属性分别

为“打开”、“关闭”、“刷新”，修改 Name 属性分别为“openSerialBtn”、“closeSerialBtn”、“refreshSerialBtn”。

（3）建立测试防冲突功能模块。在工具箱中拖出 GroupBox，并将属性窗口中的 Text 属性修改为“测试防冲突”；在“测试防冲突”的 GroupBox 中拖入两个 Button，将 Name 属性分别设为“btn_InventoryNoneAnticoll”和“btn_InventoryWithAnticoll”，以及属性分别设为“发送不使用防冲突算法的寻卡命令”和“发送使用防冲突算法的寻卡命令”。

（4）建立信息输出模块：在工具箱中拖出 ListBox 控件，放置在窗体上，选中该控件，将 Name 的属性改为“lst_TestAnticoll”；拖出 1 个 Button 控件，设置 Name 属性为“clearBtn”，Text 属性为 “清空”。

完成上述步骤后，得到的界面如图 3-49 所示。

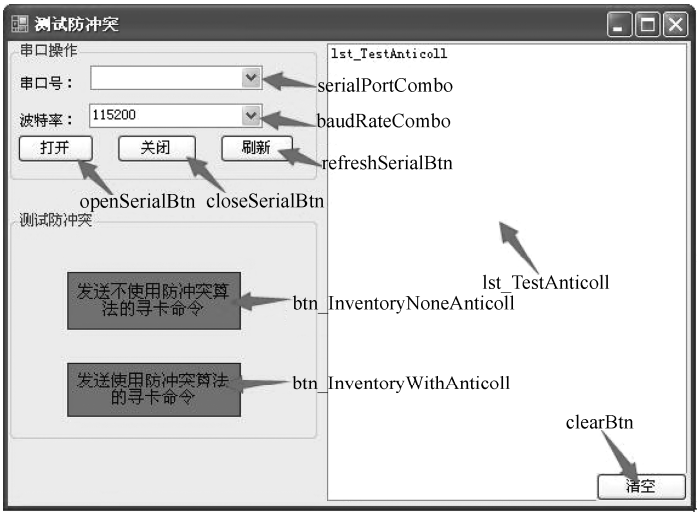


图 3-49　防冲突实验界面设计

第三步：编写代码实现功能（此处只附部分主要代码及解析）。

ISO 15693 实现防冲突算法命令及数据段如表 3-4 所示。

表 3-4　ISO 15693 实现防冲突算法命令及数据段

CMD 段	命令	Data 数据段	
13	防冲突测试	发送	1B：0 无防冲撞；1 防冲撞
		回应	UID*卡数

部分主要代码解析如下：

```
// 发送不使用防冲突算法的寻卡指令
private void btn_InventoryNoneAnticoll_Click（object sender, EventArgs e）
{
    Byte command = 0x13;                    //发送防冲突算法的寻卡命令
    Byte[] data = new Byte[1] { 0x00 };     //data 区为 00,无防冲撞
    sendOneFrame（command, isSelected, data）;//发送一帧不使用防冲突算法的寻卡命令
    AddList（"不采用防冲突算法的寻卡！"）;
    currentCMD = 0x01;                      //标记当前命令为寻卡命令
```

```
}
// 发送使用防冲突算法的寻卡指令
private void btn_InventoryWithAnticoll_Click（object sender, EventArgs e）
{
    Byte command = 0x13;                    //发送防冲突算法的寻卡命令
    Byte[] data = new Byte[1] { 0x01 };     //data 区为 01,防冲撞
    sendOneFrame（command, isSelected, data）; //发送命令
    AddList（"采用防冲突算法的寻卡！"）;
    currentCMD = 0x01;                      //当前命令为寻卡
}
```

第四步：编译生成程序运行。

编译运行程序，选择正确的串口号和波特率，单击“打开”按钮，进行发送防冲突命令测试，结果如图 3-50 所示。

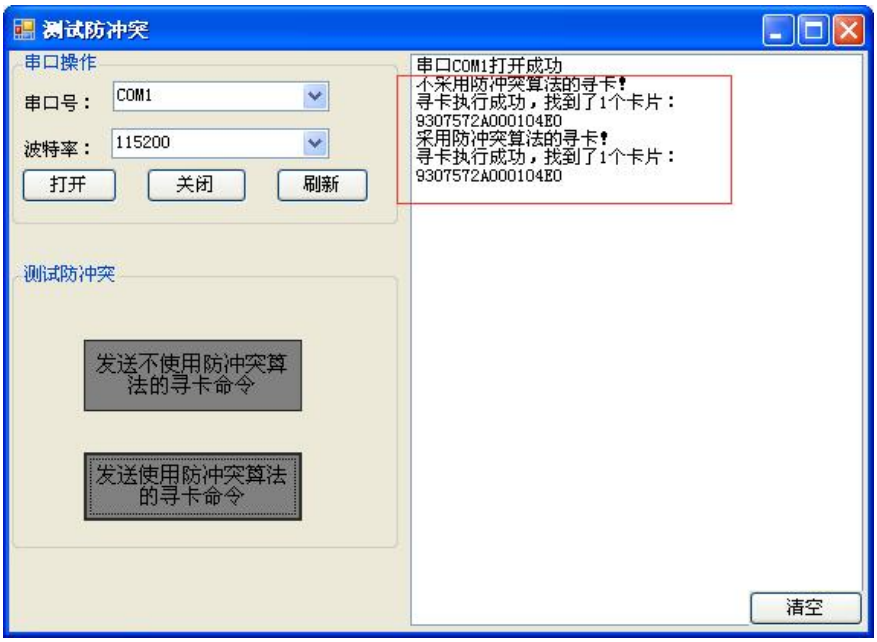


图 3-50　防冲突功能实现

第 4 章　RFID 高频 ISO 14443 协议原理及实践开发

本章所述内容所使用设备包括桂林华智 RFID 物联网教学科研平台实验箱高频 ISO 14443 模块读写器、ISO 14443 卡片、串口线。主要操作内容：使用上位机软件，与 RFID 高频 ISO 14443 模块进行通信，完成对高频 ISO 14443 标签的操作。通过本章的学习，掌握对高频 ISO 14443 模块读写器的操作，了解相关协议及工作原理。

（1）读写 IOS 14443 协议标签的读写器工作电压 5V，工作温度 0℃～60℃，工作频段是 HF 13.56MHz。这种读写器适用于多种场合读写卡应用，广泛应用于一些物品追踪、人员管理、交通罚单管理及一卡通等场合。

（2）产品电路图，如图 4-1 所示。

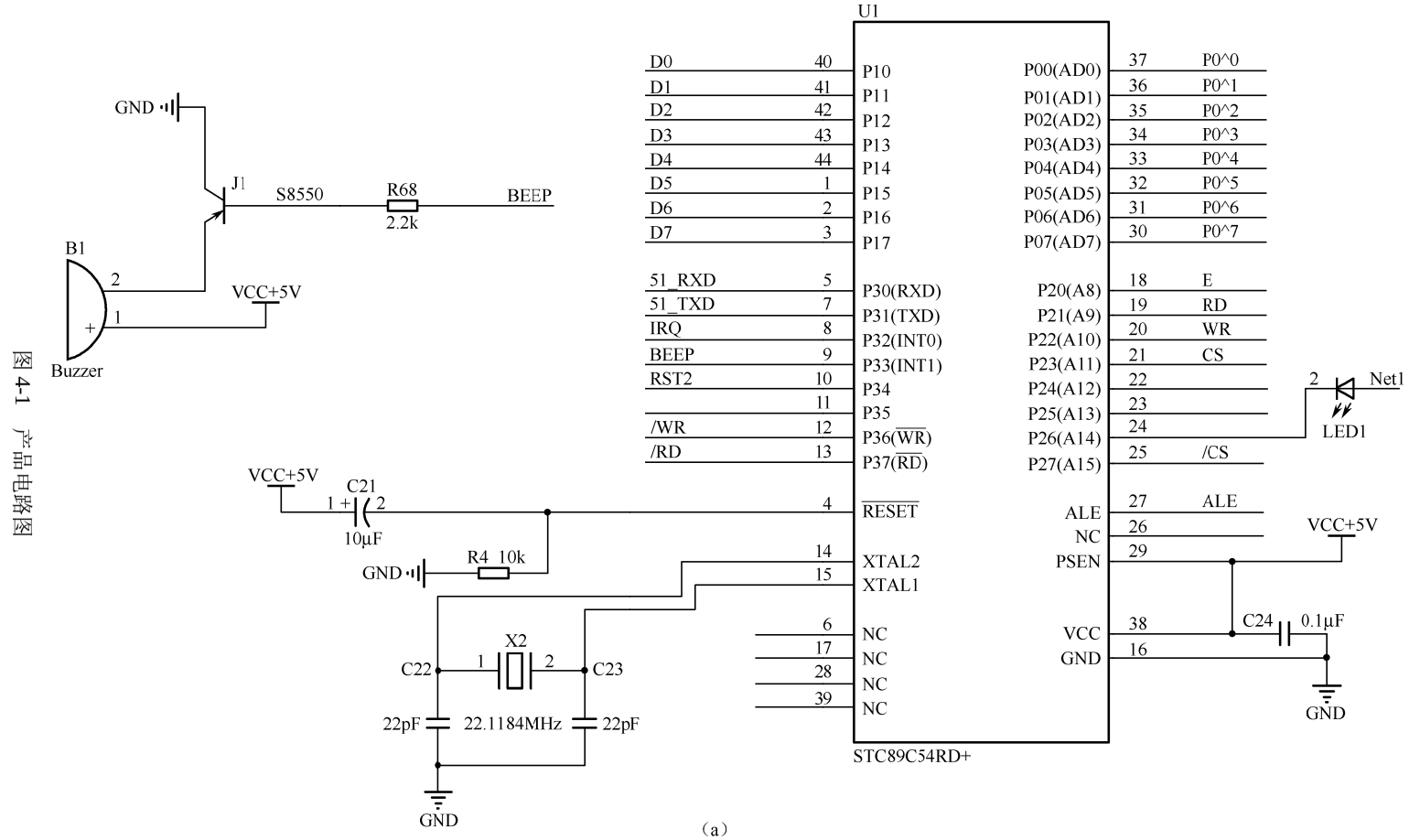
（3）产品平面图，如图 4-2 所示。

（4）工作原理简介。

MCU（微处理器 CPU）采用 STM8 芯片，驱动芯片为 MFRC531 芯片。

MFRC531 芯片的特点：高度集成的调制解调电路，可输出驱动机接至天线，支持 ISO/IEC14443A/B 和 MIFARE 经典协议；采用 Cryptol 加密算法并含有安全的非易失性内部密钥存储器；内部振荡缓存器连接 13.56MHz 石英晶体。

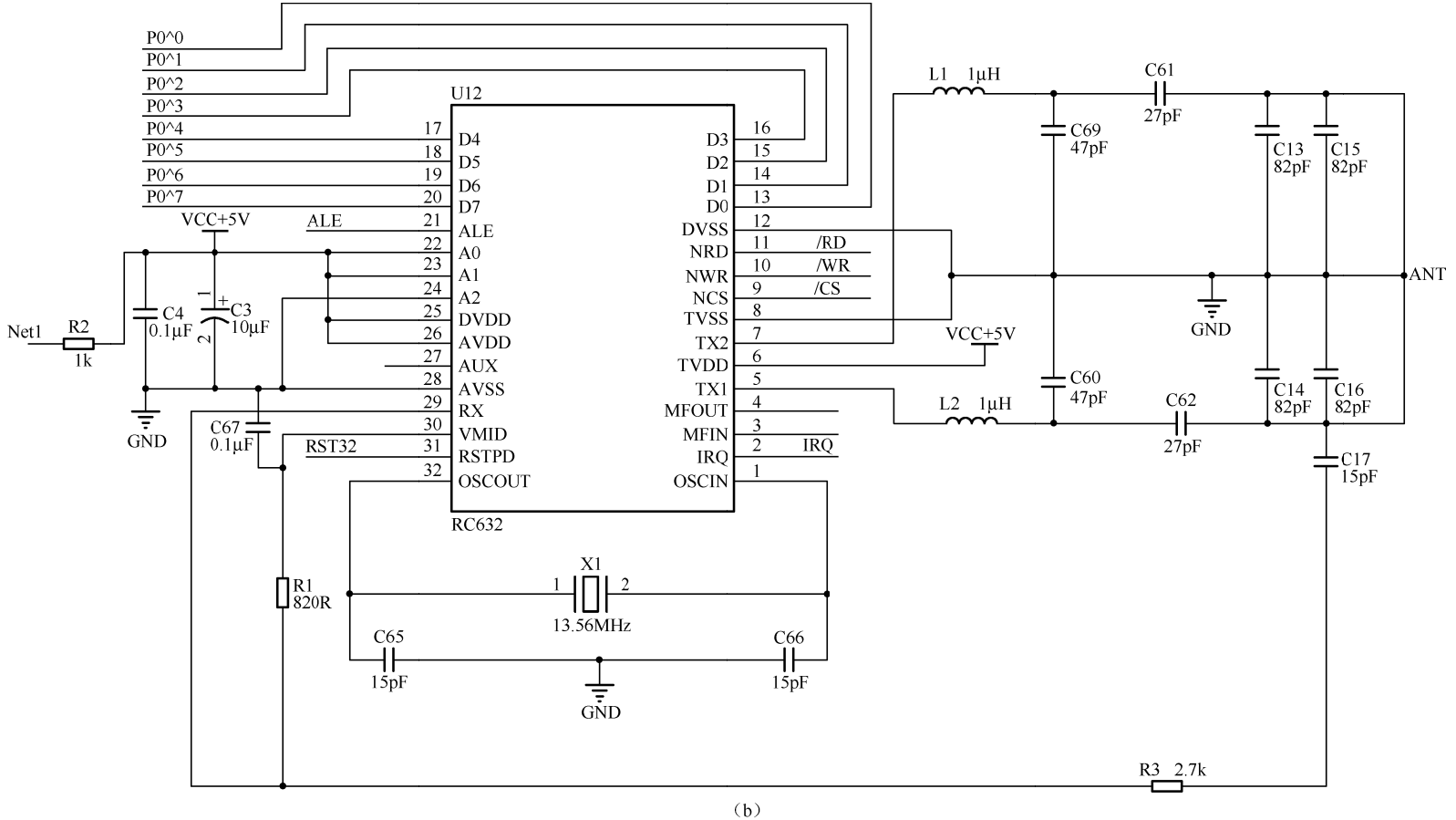
MFRC531 负责读写器对非接触式智能卡和标签的读写功能，其基本功能包括调制、解调、产生射频信号和安全管理，是读写器 MCU（微控制器）与非接触式智能卡和标签信息交换的桥梁。



（a）

图4.1　产品电路图

图 4-1　产品电路图


U12
RC632
P0^0
P0^1
P0^2
P0^3
P0^4
P0^5
P0^6
P0^7
D4
D5
D6
D7
ALE
A0
A1
A2
DVDD
AVDD
AUX
AVSS
RX
VMID
RSTPD
OSCOUT
D3
D2
D1
D0
DVSS
NRD
NWR
NCS
TVSS
TX2
TVDD
TX1
MFOUT
MFIN
IRQ
OSCIN
/RD
/WR
/CS
IRQ
VCC+5V
GND
Net1
R2
1k
C4
0.1μF
C3
10μF
C67
0.1μF
RST32
R1
820R
X1
13.56MHz
C65
15pF
C66
15pF
L1 1μH
L2 1μH
C61
27pF
C69
47pF
C60
47pF
C62
27pF
C13
82pF
C15
82pF
C14
82pF
C16
82pF
C17
15pF
ANT
R3 2.7k

（b）

图4.1　产品电路图（续）

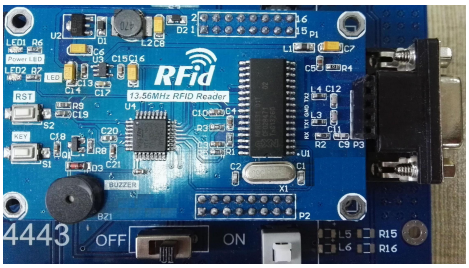


图 4-2　产品平面图

4.1　HF 13.56MHz ISO 14443 模块寻卡

4.1.1　协议原理

（1）ISO/IEC 14443 国际标准协议第三部分初始化和防碰撞算法，第四部分传输协议。

（2）上位机向下位机发送数据帧的格式如下：

SOF	CMD	协议 Type	Select	DataLength	Data	保留	EOF
2B	1B	1B	1B	2B	DataLengthByte	2B	2B

单击“请求所有”按钮，发送的数据帧：

```
EE CC 01 03 00 00 01 01 00 00 0D 0A
```

单击“寻卡”按钮，发送的数据帧：

```
EE CC 02 03 00 00 01 01 00 00 0D 0A
```

数据帧解析如下。

上述两帧数据的差异在于第三字节，第一帧数据中的“01”代表请求卡类型，第二帧数据中的“02”代表寻卡操作；每个数据帧以头两个字节 EE CC 标识数据命令开始，以最后两个字节 0D 0A 标识一帧命令结束；第四字节“03”代表操作对象为 14443 高频模块；第五字节“00”代表不在选择模式；第六、七字节“00 01”代表 Data 数据长度为 1 个字节；第八字节“01”代表 Data 数据区域数据；第九、十字节“00 00”代表保留区数据。

4.1.2　操作步骤

（1）打开上位机软件，启动 RFID 原理机的电源，使用串口线建立通信连接。

（2）读取卡片的 UID。将卡片放到识别区（如果连接了外接天线，则将卡片放置在天线旁边），在执行寻卡操作之前需要将串口（默认波特率为 115200，若改成其他值，则可能造成无法正确接收到数据）打开，打开串口成功返回如图 4-3 所示。单击“请求所有”按钮，发送“请求所有”操作请求，成功返回如图 4-4 所示的信息。

在“请求所有”成功后，如果单击“休眠”按钮，读写器将休眠，即不执行来自于上位机发送寻卡、读卡、写卡指令。此时需要单击“请求未休眠”按钮或者重新单击“请求所有”按钮。

图 4-3　打开串口

图 4-4　请求所有

（3）单击“寻卡”按钮，执行寻卡操作。寻卡成功返回如图 4-5 所示的信息。

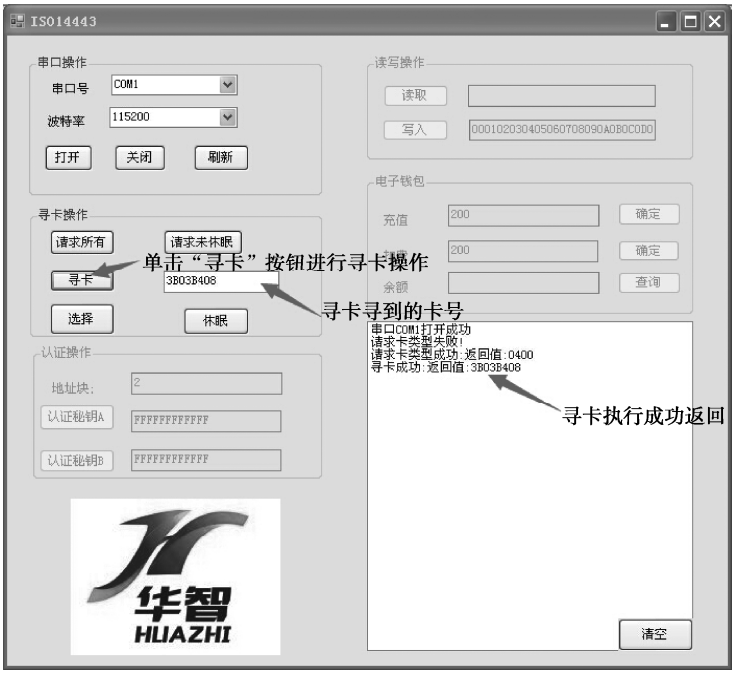


图 4-5　寻卡操作

4.1.3　程序设计：寻卡功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境下创建 C#窗体应用程序；编写代码，实现寻卡功能。主要目的是了解 Microsoft Visual Studio 2010 开发环境及窗体应用程序建立；掌握程序通过串口与 RFID 实验系统平台建立连接并通信的原理和方法；了解 RFID 实验平台 COM 协议并实现寻卡功能。

本设计所需设备如下。

（1）硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），ISO 14443（高频 13.56MHz）RFID 原理模块（基于 32 位 ARM STM32 嵌入式处理器），ISO 14443 卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计主要原理：上位机应用程序采用 C#语言开发，遵守 C#编程规范，寻卡功能根据 RFID 高频模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 RFID 高频模块连接；上位机程序根据 RFID 高频模块 COM 协议通过串口与 RFID 高频模块进行通信，最终完成寻卡的功能。

2．设计步骤

第一步：创建 C#窗体应用程序。

启动 Microsoft Visual Studio 2010 开发平台，创建一个如图 4-6 所示的 C#窗体应用程序，命名为“ISO14443_FindCard”，创建的详细方法可参考第 3 章。

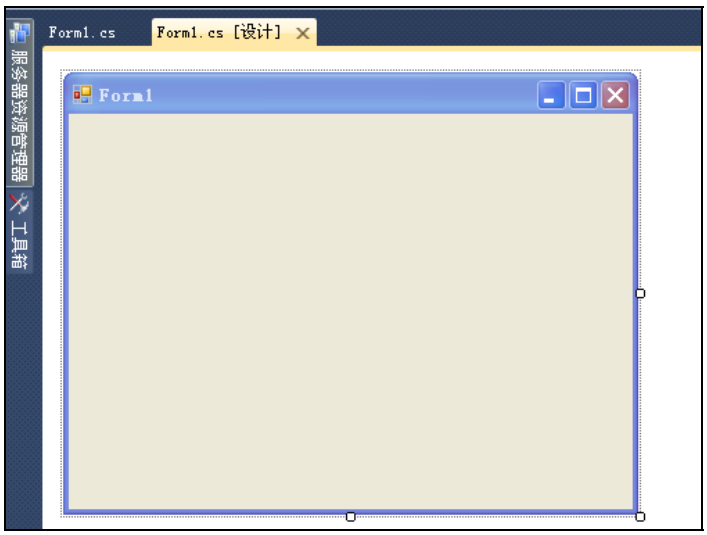


图 4-6　新建窗体

第二步：设计功能界面。

（1）选中新建的窗体，右击界面，在弹出的快捷菜单中选择“属性”选项，将属性 Text 改为“ISO14443_FindCard”。

（2）打开工具箱  ，拖出一个 GroupBox，将 Text 属性改为“串口操作”，拖出两个 Label 放到串口操作的 GroupBox 中，并将其 Text 属性分别改为 “串口号”与“波特率”，拖出两个 ComboBox 放到串口操作的 GroupBox 中，并将 ComboBox 与“串口号”对应的 Name 属性改为“PortCmb”，将 ComboBox 与“波特率”对应的 Name 属性改为“BaudRateCmb”，并将 Items 属性设为 9600、19200、57600、115200，如图 4-7 所示。

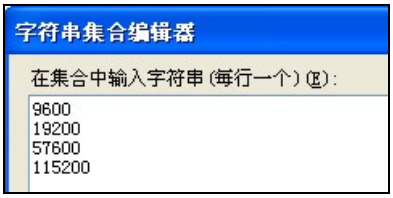


图 4-7　字符串集合编程器

将 Text 属性设置为“115200”；拖出三个 Button 放到 GroupBox 中，分别将其 Text 属性设置为“打开”、“关闭”、“刷新”，Name 属性分别对应为“OpenBtn”、“CloseBtn”、“RefreshBtn”。

（3）再拖出一个 GroupBox，将 Text 属性改为“寻卡操作”，拖出 3 个 Button 放到寻卡操作的 GroupBox 中，将其 Text 属性分别设为“请求所有”、“请求未休眠”、“寻卡”，将对应的 Name 属性设为“AllBtn”、“NoSleepBtn”、“FindBtn”；拖出一个 TextBox 控件，将 Name 属性设置为“txt_Snr”。

（4）拖出一个 ListBox 控件，将其 Name 属性设置为“receive”，拖出一个 Button 放到其中，将 Name 属性设置为“receive”，Text 属性为“清空”。

完成后的功能界面如图 4-8 所示。

第三步：编写代码实现功能（此处只附部分主要代码及解析）。

程序执行流程图如图 4-9 所示。

图 4-8　寻卡界面设计

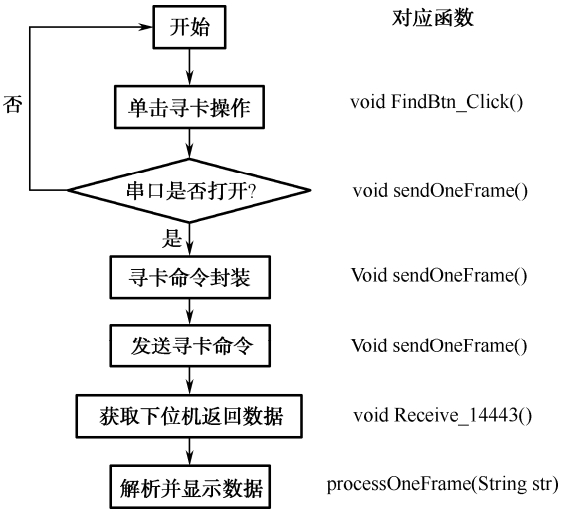


图 4-9　程序执行流程图

部分主要代码解析如下：

```
private void sendOneFrame(Byte command, Byte[] data) //发送一帧数据的方法
{
    if (!IsOpen_14443)//检查串口是否已经打开
    {
        ShowInfo(String.Format("错误：请先通过串口或网口连接设备！"));//信息显示
        HideAll_14443();//把相应的控件禁用
```

```
            return;        //如果串口没打开,直接跳出函数
        }
        Int32 dataLen = data.Length;
        Byte[] frame = new Byte[dataLen + 11];              //新建字节型数组
        frame[0] = 0xEE;       //开始帧
        frame[1] = 0xCC;       //开始帧
        frame[2] = command; //命令
        frame[3] = 0x03;       //03 代表操作对象为 14443 高频模块
        frame[4] = 0x00;       //00 代表不在选择模式
        frame[5] = (Byte)(dataLen / 256);
        frame[6] = (Byte)(dataLen % 256);                   //Data 的数据长度
        for (Int32 i = 0; i < data.Length; i++)
        {
            frame[7 + i] = data[i];                         //Data 数据
        }
        frame[7 + dataLen] = 0x00;
        frame[8 + dataLen] = 0x00;                          //保留区
        frame[9 + dataLen] = 0x0D;
        frame[10 + dataLen] = 0x0A;                         //结束标识
        try
        {
            serialPort.Write(frame, 0, frame.Length);       //向串口发送一帧数据命令
            CurrCMD_14443 = command;
        }
        catch (Exception ex)
        {
            ShowInfo(String.Format("错误：数据发送异常，错误信息为{0}", ex.Message));
            IsOpen_14443 = false;
        }
    }
    private void FindBtn_Click(object sender, EventArgs e)  //寻卡按钮单击事件
    {
        Byte[] data = new Byte[1];                          //新建一个字节长度的字节数组
        data[0] = 0x01;                                     //为字节型数组赋值
        sendOneFrame(0x02, data);                           //发送一帧寻卡命令,0x02 标明命令为寻卡命令
    }
    /*其他函数及功能(具体函数定义见源代码)
    ISO 14443 获取串口的方法:public Int32 GetComList(out List<String> ComList)  功能:获取本机串口
    打开串口方法:private void openBtn_Click(object sender, EventArgs e)    功能:打开串口
    关闭串口方法:private void closeBtn_Click(object sender, EventArgs e)  功能:关闭串口
    ISO 14443 接收数据的方法:private void Receive_14443()  功能:接收下位机发送的数据
    ISO 14443 解析一帧数据方法: private void processOneFrame(String str)  功能:对接收的数据进行解析
    */
```

第四步：编译生成程序运行

（1）在菜单中选择“调试”→“启动调试”选项或按 F5 键生成程序并运行。

（2）选择正确的波特率和串口号，单击“打开”按钮，将 ISO 14443 的电子标签放置在读卡区，串口打开成功后单击“寻卡”按钮，执行寻卡操作，结果如图 4-10 所示。

图 4-10　寻卡功能实现

4.2　HF 13.56MHz ISO 14443 模块认证

4.2.1　协议原理

（1）ISO/IEC 14443 国际标准协议第三部分初始化和防碰撞算法，第四部分传输协议。

（2）上位机向下位机发送数据帧格式如下：

SOF	CMD	协议 Type	Select	DataLength	Data	保留	EOF
2B	1B	1B	1B	2B	DataLengthByte	2B	2B

以卡号“3B03B408”为例，单击“选择”按钮，上位机向下位机发送的数据帧为:

```
EE CC 03 03 00 00 04 3B 03 B4 08 00 00 0D 0A
```

数据帧解析如下。

数据帧以头两个字节 EE CC 标识开始，以最后两个字节 0D 0A 标识结束；第三字节“03”代表选定一张卡操作；第四字节“03”代表操作对象为 14443 高频模块；第五字节“00”代表选择模式为“不在选择模式”；第六、七字节“00 04”代表 Data 数据长度为 4 个字节；第八字节“01”为 Data 数据区域数据；第九、十字节“00 00”代表保留区数据。

以卡号“3B03B408”为例，单击“认证密钥 A”按钮,上位机向下位机发送的数据帧为：

```
EE CC 04 03 00 00 0C 00 02 FF FF FF FF FF FF 3B 03 B4 08 00 00 0D 0A
```

以卡号“3B03B408”为例，单击“认证密钥 B”按钮,上位机向下位机发送的数据帧为：

```
EE CC 04 03 00 00 0C 01 02 FF FF FF FF FF FF 3B 03 B4 08 00 00 0D 0A
```

数据帧解析如下。

数据帧以头两个字节 EE CC 开始，以最后两个字节 0D 0A 结束；第三字节“04”代表认证密钥操作；第四字节“03”代表操作对象为 14443 高频模块；第五字节“00”代表选择模

式为“不在选择模式”；第六、七字节“00 0C”代表 Data 数据长度为 12 个字节;第八字节开始“00 02 **FF FF FF FF FF FF** 3B 03 B4 08”为 Data 数据区域数据（密钥 B 为 01 02 **FF FF FF FF FF FF** 3B 03 B4 08）;第二十、二十一字节“00 00”代表保留区数据。

4.2.2 操作步骤

通过使用上位机软件与 RFID 高频 ISO 14443 模块下位机进行通信，完成对高频 ISO 14443 标签的认证操作。

（1）打开上位机软件，启动 RFID 原理机的电源，用串口线建立通信连接。

（2）参照 4.1 节的步骤，读取卡片的 UID。将卡片放到识别区（如果连接了外接天线，将卡片放置在天线旁边），在执行寻卡操作之前需要将串口打开（默认波特率为 115200，若改成其他值，可能造成无法正确接收到数据），并发送“请求所有”操作请求，最后将卡号识别出来。

（3）选择操作的卡片，选择卡片成功返回如图 4-11 所示的信息。

图 4-11　成功获取卡片信息

（4）选择成功后，就可以进行认证操作。认证操作有两个认证密钥，默认密钥均为“FF FF FF FF FF FF”，上位机软件功能不包含更改认证密钥的功能。通过密钥认证操作，完成身份认证，可以对卡进行读写卡操作，以及对电子钱包进行操作。

密钥认证操作成功返回如图 4-12 所示的信息。如果将默认密钥更改，将会造成认证失败，认证失败返回如图 4-13 所示的信息。

图 4-12　认证密钥操作

图 4-13　认证密钥异常情况

4.2.3　程序设计：认证操作功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境下创建 C#窗体应用程序；编写代码，实现认证操

作功能。主要目的是了解 Microsoft Visual Studio 2010 开发环境及窗体应用程序建立；掌握程序通过串口与 RFID 实验系统平台建立连接并通信的原理和方法；了解 RFID 实验平台 COM 协议并实现认证操作功能。

本设计所需设备如下。

（1）硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），ISO 144443（高频 13.56MHz）RFID 原理模块（基于 32 位 ARM STM32 嵌入式处理器），ISO 14443 卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计主要原理：上位机应用程序采用 C#语言开发，遵守 C#编程规范，认证操作功能根据 RFID 高频模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 RFID 高频模块连接；上位机程序根据 RFID 高频模块 COM 协议通过串口与 RFID 高频模块进行通信，最终完成认证操作的功能。

2．设计步骤

第一步：创建 C#窗体应用程序。

启动 Microsoft Visual Studio 2010 开发平台，创建一个如图 4-14 所示的 C#窗体应用程序，命名为“ISO14443_Auth”，创建的详细方法可参考第 3 章。

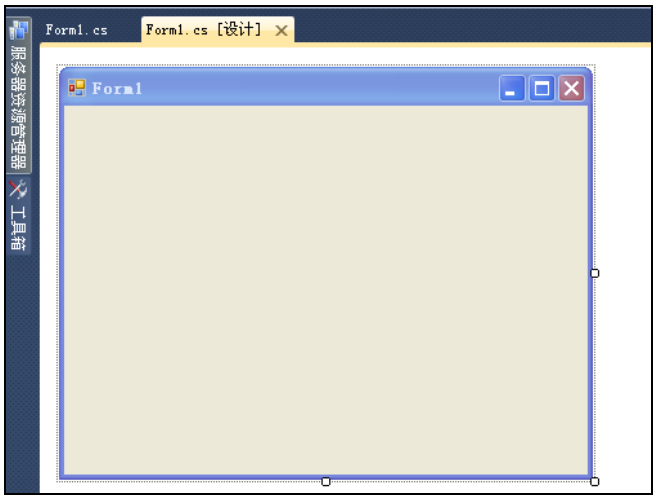


图 4-14　新建窗体

第二步：设计功能界面。

（1）选中新建的窗体，右击界面，在弹出的快捷菜中选择“属性”选项，将 Text 属性改为“ISO14443_FindCard”。

（2）打开工具箱 ，拖出一个 GroupBox，将 Text 属性改为“串口操作”，拖出两个 Label 放到串口操作的 GroupBox 中，并将其 Text 属性分别改为 “串口号”与“波特率”，拖出两个 ComboBox 放到串口操作的 GroupBox 中，并将 ComboBox 与“串口号”对应的 Name 属性改为“PortCmb”，将 ComboBox 与“波特率”对应的 Name 属性改为“BaudRateCmb”，并将 Items 属性设为 9600、19200、57600、115200，如图 4-15 所示。

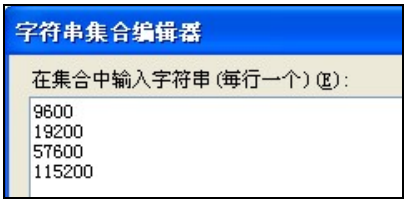


图 4-15　字符串集合编辑器

将 Text 属性设置为“115200”；拖出三个 Button 放到 GroupBox 中，分别将其 Text 属性设置为“打开”、“关闭”、“刷新”，Name 属性分别对应为“OpenBtn”、“CloseBtn”、“RefreshBtn”。

（3）再拖出一个 GroupBox，将 Text 属性改为“寻卡操作”，拖出 3 个 Button 放到寻卡操作的 GroupBox 中，将其 Text 属性分别设置为“请求所有”、“请求未休眠”、“寻卡”、“选择”，将对应的 Name 属性设置为“AllBtn”、“NoSleepBtn”、“FindBtn”、“SelectBtn”；拖出一个 TextBox 控件，将 Name 属性设置为“txt_Snr”。

（4）拖出一个 ListBox 控件，将其 Name 属性设置为“receive”，拖出一个 Button 放到其中，将 Name 属性设置为“receive”，Text 属性为“清空”。

（5）拖出一个 GroupBox，将 Text 属性改为“认证”，拖出一个 Label，将其 Text 属性改为“地址块”，拖出两个 Button，分别将 Text 属性设置为“认证密钥 A”、“认证密钥 B”，将对应 Name 属性设置为“AuthABtn”、“AuthBBtn”，拖出 3 个 TextBox 分别将其 Name 属性设置为“txt_Address”、“txt_KeyA”、“txt_KeyB”

完成后的功能界面如图 4-16 所示。

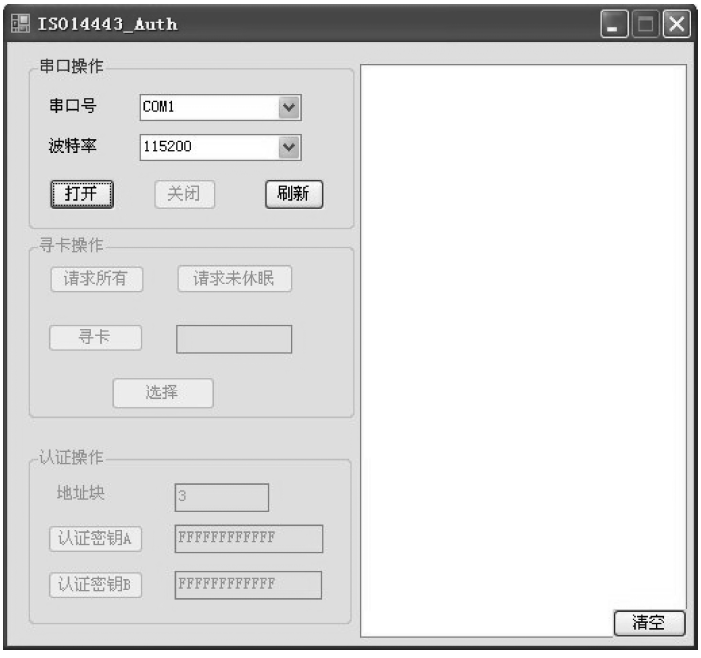


图 4-16　认证密钥界面设计

第三步：编写代码实现功能（此处只附部分主要代码及解析）。

程序执行流程图如图 4-17 所示。

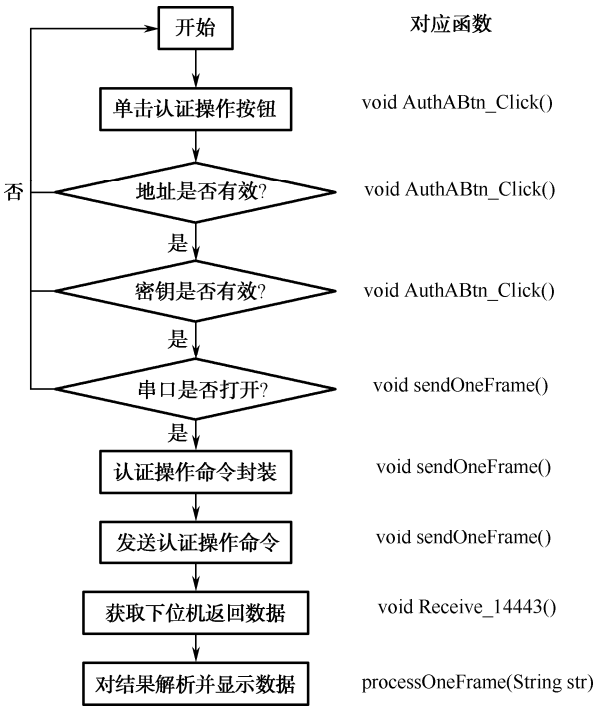


图 4-17　程序执行流程图

部分主要代码解析如下：

```
//认证操作 A 按钮事件（认证操作 B 按钮事件类似）
private void AuthABtn_Click(object sender, EventArgs e)
{
    Byte address;      //新建字节类型用于保存地址信息
    Byte.TryParse(txt_Address.Text, out address);     //将地址字符串转换成等效的字节类型
    if (address <= 0)   //判断填写的地址是否有效，无效则跳出函数
    {
        ShowInfo(String.Format("您填写的地址块{0}无效", txt_Address));
        txt_Address.Focus();
        return;
    }
    Byte[] KeyA = new Byte[6];     //新建长度为 6 的字节型数组,用于保存认证密钥
    try
    {
        for (Byte I = 0; I < 6; i++)
        {
            KeyA[i]=Convert.ToByte(txt_KeyA.Text.Trim().Substring(I * 2, 2), 16);
             //将字符串类型的认证密钥分割并保存在字节型数组中
        }
    }
    catch
    {
        ShowInfo(String.Format("您填写的密码 A{0}无效！", txt_KeyA.Text));   //密钥无效
        txt_KeyA.Focus();
```

```
            return;   //填写的密钥无效则跳出函数,不执行发送数据命令
        }
        Byte[] data = new Byte[12];                    //新建长度为 12 的字节型数组
        data[0] = 0x00;
        data[1] = Byte.Parse(txt_Address.Text);        //地址块
        for (Byte I = 0; I < 6; i++)
        {
            data[I + 2] = KeyA[i];                     //认证密钥
        }
        String CardNumString = txt_Snr.Text.Trim();  //要进行认证的标签号
        for (int I = 0; I < 4; i++)
        {
            data[I + 8] = Convert.ToByte(CardNumString.Substring(I * 2, 2), 16);
            //将标签号字符串数据类型转换并保存到字节行数组中
        }
       sendOneFrame(0x04, data);   //发送一帧认证操作命令,0x04 标明数据帧为认证操作命令
    }
    /*其他函数及功能
    其他函数声明及功能已在之前设计部分中说明(具体函数定义见源码)
    */
```

第四步：编译生成程序运行。

（1）在菜单中选择“调试”→“启动调试”选项或按 F5 键生成程序并运行。

（2）选择正确的波特率和串口号，单击“打开”按钮，将 ISO 14443 的电子标签放置在读卡区，串口打开成功后单击“寻卡”按钮，执行相应操作，结果如图 4-18 所示。

图 4-18　密钥认证功能实现

4.3　HF 13.56MHz ISO 14443 模块读取数据

4.3.1　协议原理

（1）ISO/IEC 14443 国际标准协议第三部分初始化和防碰撞算法，第四部分传输协议。

（2）上位机向下位机发送数据帧的格式如下：

SOF	CMD	协议 Type	Select	DataLength	Data	保留	EOF
2B	1B	1B	1B	2B	DataLengthByte	2B	2B

当单击“读取”按钮对卡片进行读取数据时,上位机向下位机发送的数据帧命令为:

```
EE CC 05 03 00 00 01 02 00 00 0D 0A
```

数据帧解析如下。

头两个字节“EE CC”为帧头，第三字节“05”代表读块操作，第四字节“03”代表操作对象为 14443 高频模块，第五字节“00”代表卡选择模式为“不在选择模式”，第六、七字节“00 01”代表 Data 区域的数据长度为 1 个字节，“02”为 Data 区域的数据，第九、十字节“00 00”为保留区数据，最后两个字节“0D 0A”代表数据帧结束。

4.3.2 操作步骤

通过使用上位机软件与 RFID 高频 ISO 14443 模块下位机进行通信，完成对高频 ISO 14443 标签的读取数据。

（1）打开上位机软件，启动 RFID 原理机的电源，用串口线建立通信连接。

（2）参照 4.1 节的操作步骤，读取卡片的 UID。将卡片放到识别区（如果连接了外接天线，将卡片放置在天线旁边），在执行寻卡操作之前需要将串口打开（默认波特率为 115200，若改成其他值，可能造成无法正确接收到数据），并发送“请求所有”操作请求，然后完成寻卡操作。

（3）参照 4.2 节的操作步骤完成认证操作。

（4）选择要读取的卡片，单击“读取”按钮，如图 4-19 所示。

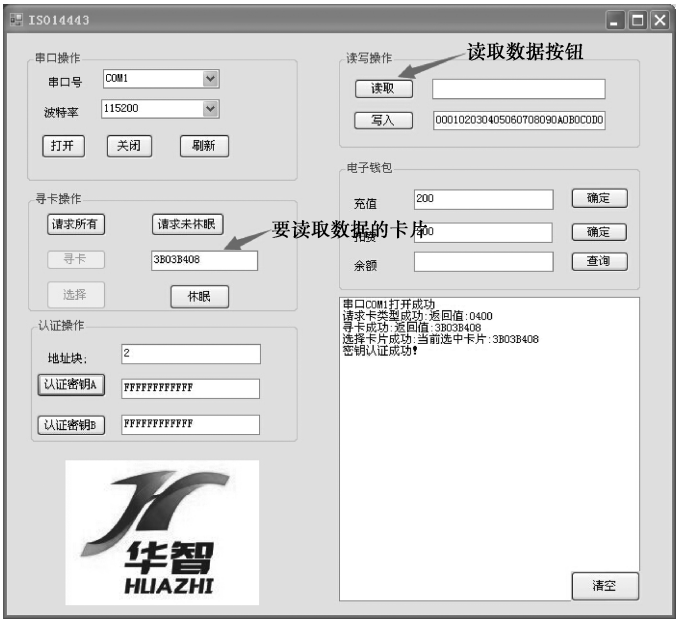


图 4-19　数据读取功能操作

寻卡成功返回如图 4-20 所示的信息。如果在读数据过程中将卡片从读卡区移走，下位机将无法寻卡，就会返回寻卡失败。寻卡失败返回如图 4-21 所示的信息。

图 4-20　成功读取数据

图 4-21　数据读取失败情况

4.3.3　程序设计：读取数据功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境下创建 C#窗体应用程序；编写代码，实现读取数据功能。主要目的是了解 Microsoft Visual Studio 2010 开发环境及窗体应用程序建立；掌握程

序通过串口与 RFID 实验系统平台建立连接并通信的原理和方法；了解 RFID 实验平台 COM 协议并实现读取数据功能。

本设计所需设备如下。

（1）硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），ISO 14443（高频 13.56MHz）RFID 原理模块（基于 32 位 ARM STM32 嵌入式处理器），ISO 14443 卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计主要原理：上位机应用程序采用 C#语言开发，遵守 C#编程规范，读取数据功能根据 RFID 高频模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 RFID 高频模块连接；上位机程序根据 RFID 高频模块 COM 协议通过串口与 RFID 高频模块进行通信，最终完成读取数据的功能。

2．设计步骤

第一步：创建 C#窗体应用程序。

启动 Microsoft Visual Studio 2010 开发平台，创建一个如图 4-22 所示的 C#窗体应用程序，命名为“ISO14443_ Read”，创建的详细方法可参考第 3 章。

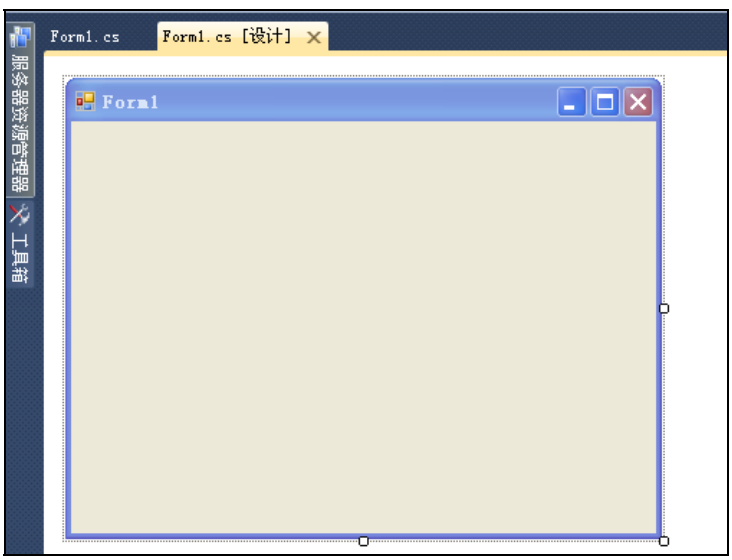


图 4-22　新建窗体

第二步：设计功能界面。

（1）选中新建的窗体，右击界面，在弹出的快捷菜单中选择“属性”选项，将 Text 属性改为“ISO14443_Read”。

（2）打开工具箱 ，拖出一个 GroupBox，将 Text 属性改为“串口操作”，拖出两个 Label 放到串口操作的 GroupBox 中，并将其 Text 属性分别改为 “串口号”与“波特率”，拖出两个 ComboBox 放到串口操作的 GroupBox 中，并将 ComboBox 与“串口号”对应的 Name 属性改为“PortCmb”，将 ComboBox 与“波特率”对应的 Name 属性改为“BaudRateCmb”，并将 Items 属性设置为 9600、19200、57600、115200，如图 4-23 所示。

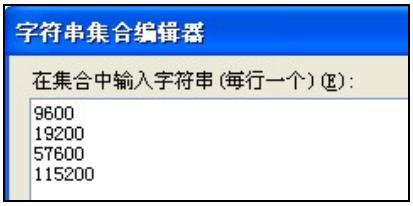


图 4-23　字符串集合编辑器

将 Text 属性设置为“115200”；拖出三个 Button 放到 GroupBox 中，分别将其 Text 属性设置为“打开”、“关闭”、“刷新”，Name 属性分别对应为“OpenBtn”、“CloseBtn”、“RefreshBtn”。

（3）再拖出一个 GroupBox，将 Text 属性改为“寻卡操作”，拖出 3 个 Button 放到寻卡操作的 GroupBox 中，将其 Text 属性分别设置为“请求所有”、“请求未休眠”、“寻卡”、“选择”，将对应的 Name 属性设置为“AllBtn”、“NoSleepBtn”、“FindBtn”、“SelectBtn”；拖出一个 TextBox 控件，将 Name 属性设置为“txt_Snr”。

（4）拖出一个 ListBox 控件，将其 Name 属性设置为“receive”，拖出一个 Button 放到其中，将 Name 属性设置为“receive”，Text 属性为“清空”。

（5）拖出一个 GroupBox，将 Text 属性改为“认证”，拖出一个 Label，将其 Text 属性改为“地址块”，拖出两个 Button，分别将 Text 属性设置为“认证密钥 A”、“认证密钥 B”，将对应 Name 属性设置为“AuthABtn”、“AuthBBtn”，拖出 3 个 TextBox 分别将其 Name 属性设为“txt_Address”、“txt_KeyA”、“txt_KeyB”。

（6）拖出一个 GroupBox，将 Text 属性改为“读写操作”，拖出一个 Button，将 Text 属性设为“读取”，将 Name 属性设置为“ReadBtn”，拖出一个 TextBox，将 Name 属性设置为“txt_DataOut”。

完成后的功能界面如图 4-24 所示。

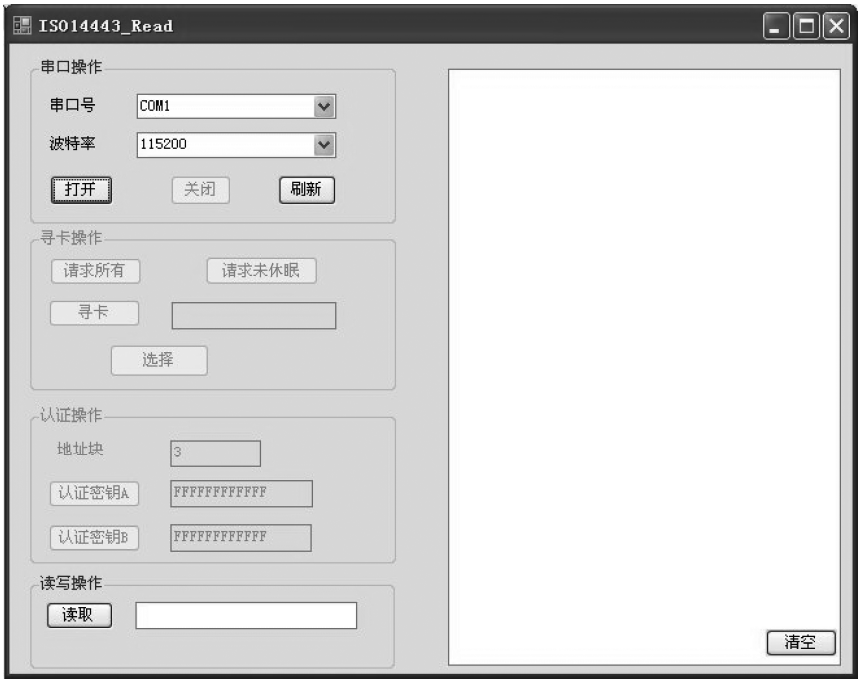


图 4-24　读取数据界面

第三步：编写代码实现功能（此处只附部分主要代码及解析）

程序执行流程图如图 4-25 所示。

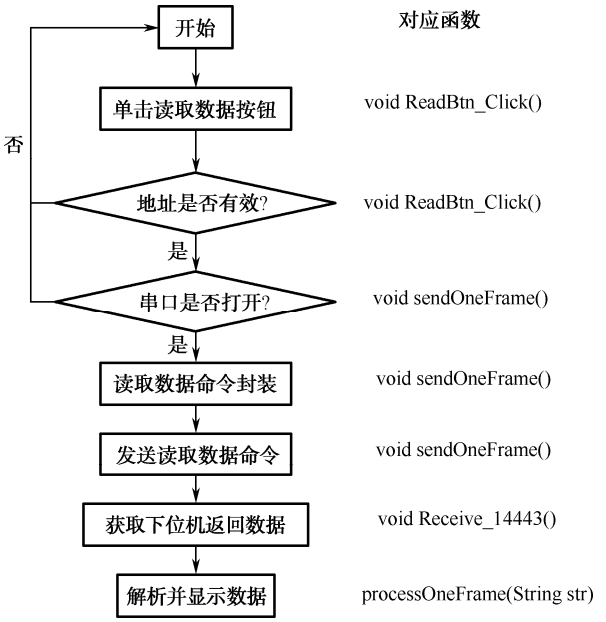


图 4-25　程序执行流程图

部分主要代码解析如下：

```
//读取按钮事件
private void ReadBtn_Click(object sender, EventArgs e)
{
    address;                    //新建字节类型变量保存填写的地址块信息
    Byte.TryParse(txt_Address.Text, out address);
                                //将字符串类型的地址块数据类型转换
    if (address <= 0)           //判断填写的地址是否有效,无效则跳出函数
    {
        ShowInfo(String.Format("您填写的地址块{0}无效！", txt_Address));
        txt_Address.Focus();
        return;
    }
    Byte[] data = new Byte[1];    //字节型数组,用于保存填写的地址块信息
    data[0] = Byte.Parse(txt_Address.Text);     //赋值
    sendOneFrame(0x05, data);   //发送一帧读取数据命令,0x05 标明命令为读卡命令
    }
/*其他函数及功能
其他函数声明及功能已在之前设计部分中说明(具体函数定义见源码)
*/
```

第四步：编译生成程序运行。

（1）在菜单中选择“调试”→“启动调试”选项或按 F5 键生成程序并运行。

（2）选择正确的波特率和串口号，单击“打开”按钮，将 ISO 14443 的电子标签放置在读

卡区，串口打开成功后单击“寻卡”按钮，执行相应操作，结果如图 4-26 所示。

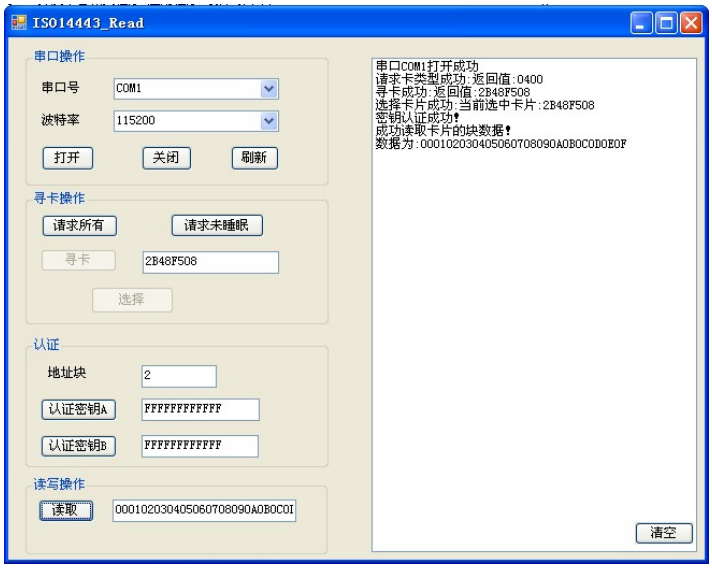


图 4-26　读取数据功能实现

4.4　HF 13.56MHz ISO 14443 模块写入数据

4.4.1　协议原理

（1）ISO/IEC 14443 国际标准协议第三部分初始化和防碰撞算法，第四部分传输协议。

（2）上位机向下位机发送数据帧的格式如下：

SOF	CMD	协议 Type	Select	DataLength	Data	保留	EOF
2B	1B	1B	1B	2B	DataLengthByte	2B	2B

当单击“写入”按钮对卡片进行写入数据时,上位机向下位机发送的数据帧命令为（写入的数据为 000102030405060708090A0B0C0D0E0F，卡号为 3B03B408）：

```
EE CC 06 03 00 00 11 02 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 00 00 0D 0A
```

数据帧解析如下。

头两个字节“EE CC”为帧头，第三字节“06”代表写入操作,第四字节“03”代表操作对象为“14443”高频模块，第五字节“00”代表卡选择模式为“不在选择模式”，第六、七字节“00 11”代表 Data 区域的数据长度为 16 个字节，写入的数据“00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F”为 Data 区域的数据，第二十五、二十六字节“00 00”为保留区数据，最后两个字节“0D 0A”代表数据帧结束。

4.4.2　操作步骤

（1）打开上位机软件，再启动 RFID 原理机的电源，用串口线建立通信连接。

（2）参照 4.1 节的操作步骤，读取卡片的 UID。将卡片放到识别区（如果连接了外接天线，将卡片放置在天线旁边），在执行寻卡操作之前需要将串口（默认波特率为 115200，若改成其

他值，可能造成无法正确接收到数据）打开，并发送“请求所有”操作请求，然后完成寻卡操作。

（3）参照 4.2 节的操作步骤完成认证操作。

（4）写入数据操作，写入成功返回如图 4-27 所示的信息。

图 4-27　写入数据操作

可能导致写入数据失败的以下情况。① 写入的数据字节数不够，写卡失败返回提示填写的数据无效，如图 4-28 所示。

图 4-28　写入失败情况

② 在写入的过程中将卡片从读写区移走，即使数据字节长度足够也将导致写入数据失败，写入数据失败返回如图 4-29 所示的信息。

图 4-29　写入数据失败情况

4.4.3　程序设计：写入数据功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境下创建 C#窗体应用程序；编写代码，实现写入数据功能。主要目的是了解 Microsoft Visual Studio 2010 开发环境及窗体应用程序建立；掌握程序通过串口与 RFID 实验系统平台建立连接并通信的原理和方法；了解 RFID 实验平台 COM 协议并实现写入数据功能。

本设计所需设备如下。

（1）硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），ISO 14443（高频 13.56MHz）RFID 原理模块（基于 32 位 ARM STM32 嵌入式处理器），ISO 14443 卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计主要原理：上位机应用程序采用 C#语言开发，遵守 C#编程规范，写入数据功能根据 RFID 高频模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 RFID 高频模块连接；上位机程序根据 RFID 高频模块 COM 协议通过串口与 RFID 高频模块进行通信，最终完成写入数据的功能。

2．设计步骤

第一步：创建 C#窗体应用程序。

启动 Microsoft Visual Studio 2010 开发平台，创建一个 C#窗体应用程序，命名为“ISO14443_Write”，创建的详细方法可参考第 3 章，如图 4-30 所示。

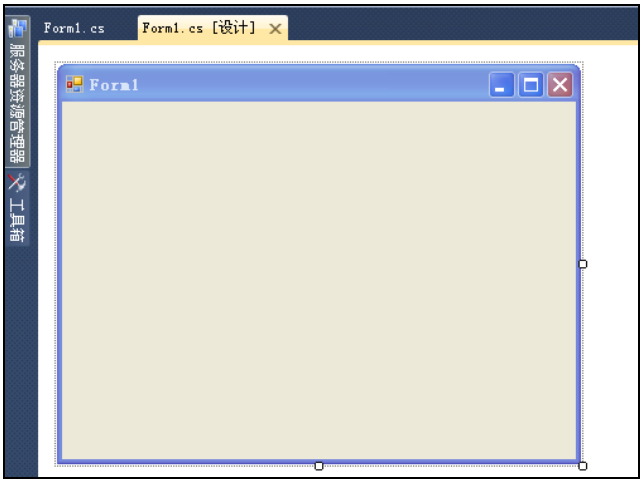


图 4-30　新建窗体

第二步：设计功能界面。

（1）选中新建的窗体，右击界面，在弹出的快捷菜单中选择“属性”选项，将 Text 属性改为“ISO 14443_ Write”。

（2）打开工具箱 ，拖出一个 GroupBox，将 Text 属性改为“串口操作”，拖出两个 Label 放到串口操作的 GroupBox 中，并将其 Text 属性分别改为 “串口号”与“波特率”，拖出两个 ComboBox 放到串口操作的 GroupBox 中，并将 ComboBox 与“串口号”对应的 Name 属性改为“PortCmb”，将 ComboBox 与“波特率”对应的 Name 属性改为“BaudRateCmb”，并将 Items 属性设置为 9600、19200、57600、115200，如图 4-31 所示。

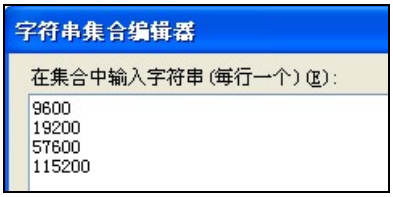


图 4-31　字符串集合编辑器

将 Text 属性设置为“115200”；拖出三个 Button 放到 GroupBox 中，分别将其 Text 属性设置为“打开”、“关闭”、“刷新”，Name 属性分别对应为“OpenBtn”、“CloseBtn”、“RefreshBtn”。

（3）再拖出一个 GroupBox，将 Text 属性改为“寻卡操作”，拖出 3 个 Button 放到寻卡操作的 GroupBox 中，将其 Text 属性分别设置为“请求所有”、“请求未休眠”、“寻卡”、“选择”，将对应的 Name 属性设置为“AllBtn”、“NoSleepBtn”、“FindBtn”、“SelectBtn”；拖出一个 TextBox 控件，将 Name 属性设置为“txt_Snr”。

（4）拖出一个 ListBox 控件，将其 Name 属性设置为“receive”，拖出一个 Button 放到其中，将 Name 属性设置为“receive”，Text 属性为“清空”。

（5）拖出一个 GroupBox，将 Text 属性改为“认证”，拖出一个 Label，将其 Text 属性改为“地址块”，拖出两个 Button，分别将 Text 属性设置为“认证密钥 A”、“认证密钥 B”，将对应 Name 属性设置为“AuthABtn”、“AuthBBtn”，拖出 3 个 TextBox 分别将其 Name 属性设置为“txt_Address”、“txt_KeyA”、“txt_KeyB”。

（6）拖出一个 GroupBox，将 Text 属性改为“读写操作”，拖出两个 Button，将 Text 属性分别设置为“读取”与“写入”，将 Name 属性设置为“ReadBtn”和“WriteBtn”，拖出两个 TextBox，将 Name 属性设置为“txt_DataOut”与“txt_DataIn”。

完成后的功能界面如图 4-32 所示。

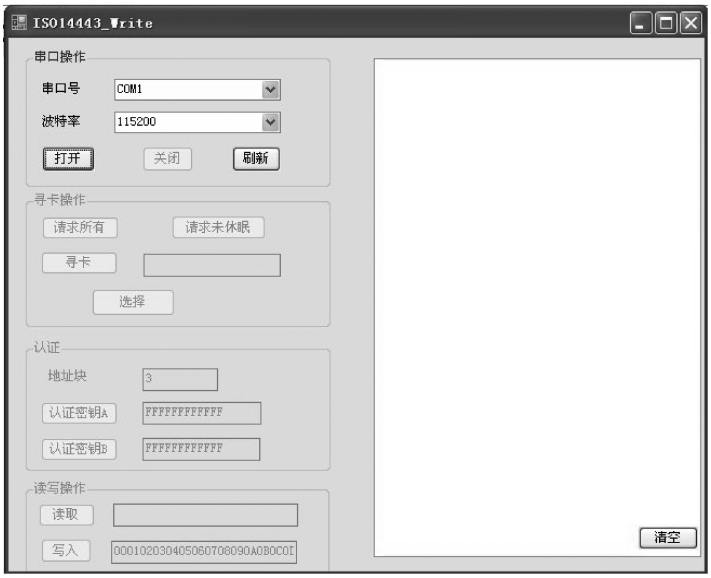


图 4-32　写入数据界面

第三步：编写代码实现功能（此处只附部分主要代码及解析）。

程序执行流程图如图 4-33 所示。

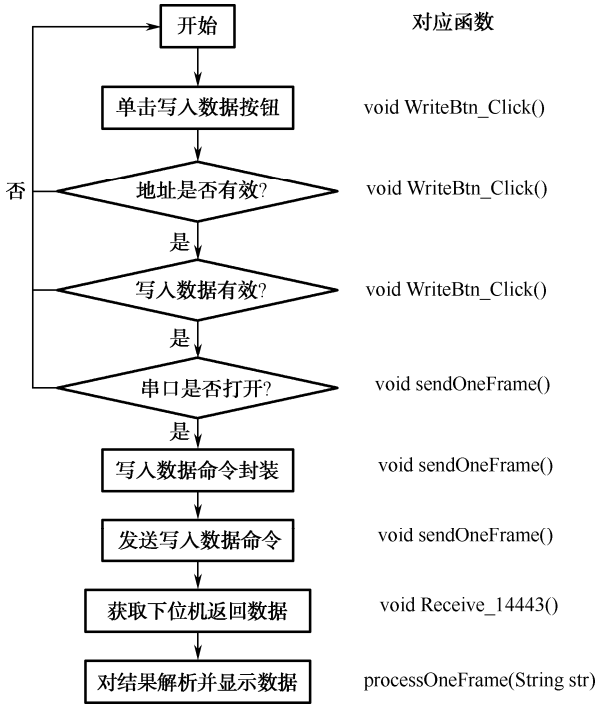


图 4-33　程序执行流程图

部分主要代码解析如下：

```
private void WriteBtn_Click(object sender, EventArgs e)     //写操作按钮事件
{
    Byte address;     //字节型变量,用于保存填写的地址块信息
    try
    {
        address = Byte.Parse(txt_Address.Text);    //强制类型转换,将字符串型转成字节型
        if (address == 0)
            throw (new Exception("0 块用于存放厂商代码，已经固化，不能被改写！"));
        if (address == 1)
            throw (new Exception("1 块用作电子钱包，受保护，不能随意改写！"));
        if ((address % 4) == 3)
            throw (new Exception("每个扇区的块 3 为控制块，为防止误操作而使扇区失效，拒绝
                改写块 3！"));   //判断填写的地址块是否合法
    }
    catch (Exception ex)                                    //填写地址块非法则跳出函数
    {
        ShowInfo(String.Format("您填写的地址块{0}无效！原因是:{1}", txt_Address.Text,
                ex.Message));
        txt_Address.Focus();
    return;
    }
    String str_data = txt_DataIn.Text.Trim();   //字符串变量,保存写入的数据
    Byte[] data_Write = new Byte[16];           //字节型数组,用于保存写入的数据
    try
    {
        for (Byte i = 0; i < 16; i++)
        {
            data_Write[i]=(Byte)(Convert.ToByte(str_data.Substring(i * 2, 2), 16));
            //将写入的字符串数据分割并进行数据类型转换保存在字节数组中
        }
    }
    catch       //填写的数据无效则跳出函数
    {
        ShowInfo("您填写的数据无效！");
        txt_DataIn.Focus();
        return;
    }
    Byte[] data = new Byte[17];
    //字节型数组,第一字节保存地址块信息,其他字节保存写入的数据
    data[0] = Byte.Parse(txt_Address.Text);
    for (Byte i = 0; i < 16; i++)
    {
        data[i + 1]=(Byte)(Convert.ToByte(str_data.Substring(i * 2, 2), 16));
        //将写入的数据保存到字节数组中
    }
    sendOneFrame(0x06, data);     //发送一帧写入命令,0x06 标明命令为写入命令
```

```
}
/*其他函数及功能
其他函数声明及功能已在本章其他设计部分中说明(具体函数定义见源码)
*/
```

第四步：编译生成程序运行。

（1）在菜单中选择“调试”→“启动调试”选项或按 F5 键生成程序并运行。

（2）选择正确的波特率和串口号，单击“打开”按钮，将 ISO 14443 的电子标签放置在读卡区，串口打开成功后单击“寻卡”按钮，执行相应操作，结果如图 4-34 所示。

图 4-34　写入数据功能实现

4.5　HF 13.56MHz ISO 14443 模块休眠

4.5.1　协议原理

（1）ISO/IEC 14443 国际标准协议第三部分初始化和防碰撞算法，第四部分传输协议。

（2）上位机向下位机发送数据帧的格式如下：

SOF	CMD	协议 Type	select	DataLength	Data	保留	EOF
2B	1B	1B	1B	2B	DataLengthByte	2B	2B

当单击“休眠”按钮进行休眠时,上位机向下位机发送的数据帧命令为：

```
EE CC 07 03 00 00 01 00 00 00 0D 0A
```

数据帧解析如下。

头两个字节“EE CC”为帧头，第三字节“07”代表休眠操作，第四字节“03”代表操作对象为 ISO 14443 高频模块，第五字节“00”代表卡选择模式为“不在选择模式”，第六、七字节“00 01”代表 Data 区域的数据长度为 1 个字节，第八字节“00”为 Data 区域的数据，第九、十字节“00 00”为保留区数据，最后两个字节“0D 0A”代表数据帧结束。

当单击“请求未休眠”按钮进行未休眠时，上位机向下位机发送的数据帧命令为：

```
EE CC 01 03 00 00 01 00 00 00 0D 0A
```

数据帧解析如下。

头两个字节“EE CC”为帧头，第三字节“01”代表请求所有，第四字节“03”代表操作对象为 ISO 14443 高频模块，第五字节“00”代表卡选择模式为“不在选择模式”，第六、七字节“00 01”代表 Data 区域的数据长度为 1 个字节，第八字节“00”为 Data 区域的数据，第九、十字节“00 00”为保留区数据，最后两个字节“0D 0A”代表数据帧结束。

4.5.2 操作步骤

（1）打开上位机软件，启动 RFID 原理机的电源，用串口线建立通信连接。

（2）参照 4.1 节的操作步骤，读取卡片的 UID。将卡片放到识别区（如果连接了外接天线，将卡片放置在天线旁边），在执行寻卡操作之前需要将串口（默认波特率为 115200，若改成其他值，可能造成无法正确接收到数据）打开。

（3）进入休眠状态，有无卡片在读卡区不影响指令的执行成功，单击“休眠”按钮，进入休眠状态，成功返回如图 4-35 所示的信息。

图 4-35　模块休眠操作

执行“请求未休眠”命令需要有卡片在读卡区，请求未休眠成功返回如图 4-36 所示的信息。如果在读卡区没有放置卡片将导致命令执行失败，命令执行失败返回如图 4-37 所示的信息。

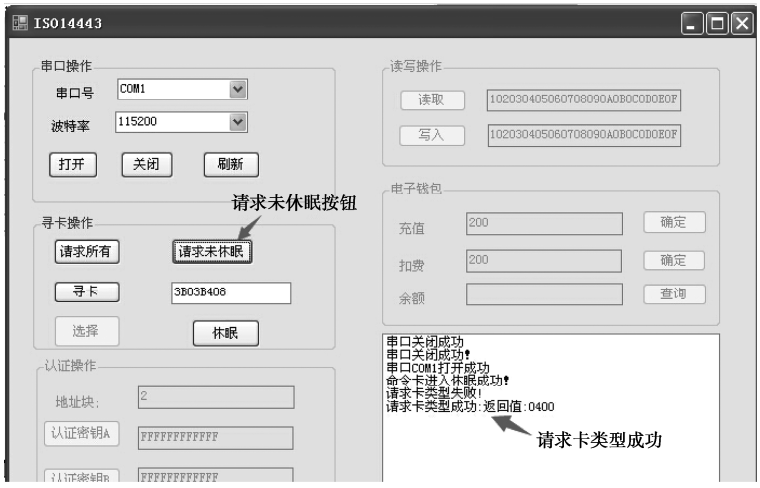


图 4-36　请求未休眠

图 4-37　请求失败情况

4.5.3　程序设计：休眠功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境下创建 C#窗体应用程序；编写代码，实现休眠功能。主要目的是了解 Microsoft Visual Studio 2010 开发环境及窗体应用程序建立；掌握程序通过串口与 RFID 实验系统平台建立连接并通信的原理和方法；了解 RFID 实验平台 COM 协议并实现休眠功能。

本设计所需设备如下。

（1）硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），ISO 14443（高频 13.56MHz）RFID 原理模块（基于 32 位 ARM STM32 嵌入式处理器），ISO 14443 卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计主要原理：上位机应用程序采用 C#语言开发，遵守 C#编程规范，休眠功能根据 RFID 高频模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 RFID 高频模块连接；上位机程序根据 RFID 高频模块 COM 协议通过串口与 RFID 高频模块进行通信，最终完成休眠的功能。

2．设计步骤

第一步：创建 C#窗体应用程序。

启动 Microsoft Visual Studio 2010 开发平台，创建一个如图 4-38 所示的 C#窗体应用程序，命名为“ISO14443_ Sleep”，创建的详细方法可参考第 3 章。

第二步：设计功能界面。

（1）选中新建的窗体，右击界面，在弹出的快捷菜单中选择“属性”选项，将 Text 属性改为“ISO14443_ Sleep”。

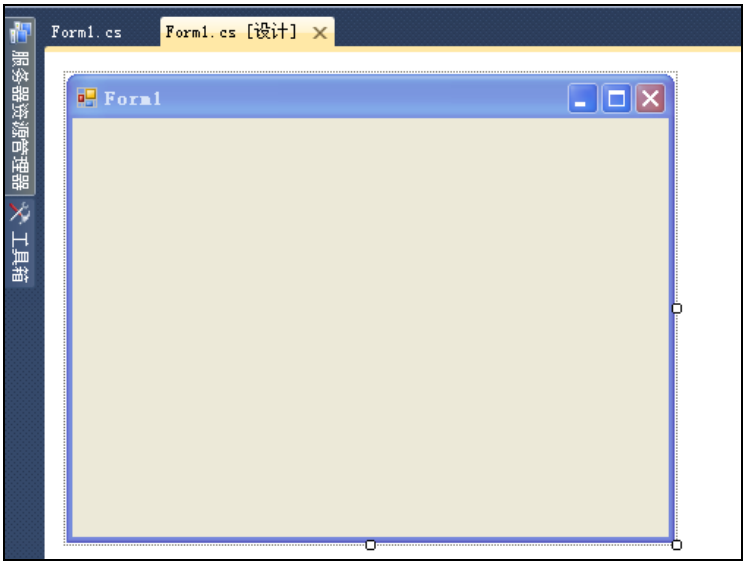


图 4-38　新建窗体

（2）打开工具箱 ，拖出一个 GroupBox，将 Text 属性改为“串口操作”，拖出两个 Label 放到串口操作的 GroupBox 中，并将其 Text 属性分别改为 “串口号”与“波特率”，拖出两个 ComboBox 放到串口操作的 GroupBox 中，并将 ComboBox 与“串口号”对应的 Name 属性改为“PortCmb”，将 ComboBox 与“波特率”对应的 Name 属性改为“BaudRateCmb”，并将 Items 属性设置为 9600、19200、57600、115200，如图 4-39 所示。

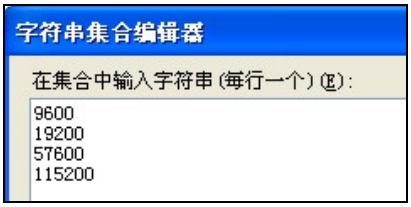


图 4-39　字符串集合编辑器

将 Text 属性设置为“115200”；拖出三个 Button 放到 GroupBox 中，分别将其 Text 属性设置为“打开”、“关闭”、“刷新”，Name 属性分别对应为“OpenBtn”、“CloseBtn”、“RefreshBtn”。

（3）再拖出一个 GroupBox，将 Text 属性改为“寻卡操作”，拖出 3 个 Button 放到寻卡操作的 GroupBox 中，将其 Text 属性分别设置为“请求所有”、“请求未休眠”、“寻卡”、“选择”，将对应的 Name 属性设置为“AllBtn”、“NoSleepBtn”、“FindBtn”、“SelectBtn”；拖出一个 TextBox 控件，将 Name 属性设置为“txt_Snr”。

（4）拖出一个 ListBox 控件，将其 Name 属性设置为“receive”，拖出一个 Button 放到其中，将 Name 属性设置为“receive”，Text 属性为“清空”。

（5）拖出一个 GroupBox，将 Text 属性改为“认证”，拖出一个 Label，将其 Text 属性改为“地址块”，拖出两个 Button，分别将 Text 属性设置为“认证密钥 A”、“认证密钥 B”，将对应的 Name 属性设置为“AuthABtn”、“AuthBBtn”，拖出 3 个 TextBox 分别将其 Name 属性设置为“txt_Address”、“txt_KeyA”、“txt_KeyB”。

（6）拖出一个 GroupBox，将 Text 属性改为“读写操作”，拖出两个 Button，将 Text 属性

分别设置为“读取”与“写入”，将 Name 属性设置为“ReadBtn”和“WriteBtn”，拖出两个 TextBox，将 Name 属性设置为“txt_DataOut”与“txt_DataIn”。再拖出一个 Button，将 Text 属性设置为“休眠”，Name 的属性“SleepBtn”。

完成后的功能界面如图 4-40 所示。

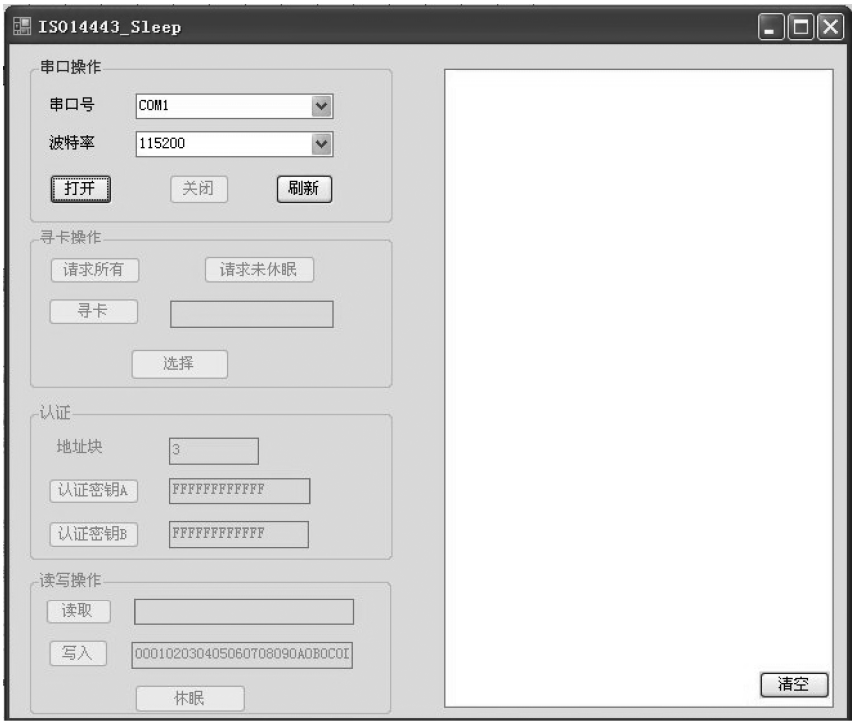


图 4-40　休眠功能界面设计

第三步：编写代码实现功能。

程序执行流程图如图 4-41 所示。

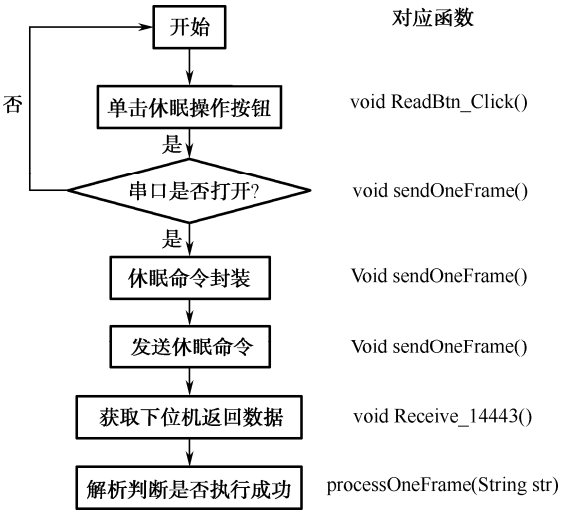


图 4-41　程序执行流程图

部分主要代码解析如下：

```
// 休眠按钮事件
private void SleepBtn_Click(object sender, EventArgs e)
{
    Byte[] data = new Byte[1]; //Data 区域数据
    data[0] = 0x00;//为 Data 区域数据赋值
    sendOneFrame(0x07, data);//发送一帧命令,0x07 标明命令为进入休眠命令
}
/*其他函数及功能
其他函数声明及功能已在第 4 章其他设计部分中说明(具体函数定义见源码)
*/
```

第四步：编译生成程序运行。

（1）在菜单中选择“调试”→“启动调试”选项或按 F5 键生成程序并运行。

（2）选择正确的波特率和串口号，单击“打开”按钮，将 ISO 14443 的电子标签放置在读卡区，串口打开成功后单击“寻卡”按钮，执行相应操作，结果如图 4-42 所示。

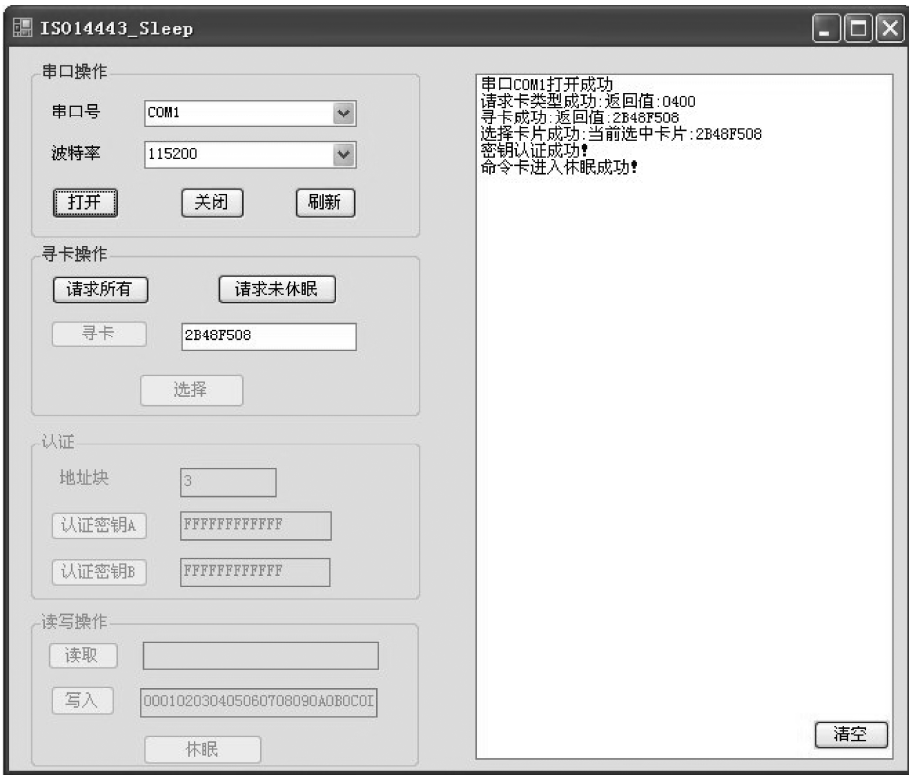


图 4-42　休眠操作成功

4.6　HF 13.56MHz ISO 14443 模块电子钱包

4.6.1　协议原理

（1）ISO/IEC 14443 国际标准协议第三部分初始化和防碰撞算法，第四部分传输协议。

（2）上位机向下位机发送数据帧的格式如下：

SOF	CMD	协议 Type	Select	DataLength	Data	保留	EOF
2B	1B	1B	1B	2B	DataLengthByte	2B	2B

当单击“充值”按钮进行电子钱包充值时（以充值金额为 200 为例），上位机向下位机发送的数据帧命令为：

```
EE CC 08 03 00 00 04 C8 00 00 00 00 00 0D 0A
```

数据帧解析如下。

头两个字节“EE CC”为帧头，第三字节“08”代表电子钱包充值，第四字节“03”代表操作对象为 ISO 14443 高频模块，第五字节“00”代表卡选择模式为“不在选择模式”，第六、七字节“00 04”代表 Data 区域的数据长度为 4 个字节，第八字节开始的 4 个字节代表钱数，即“C8 00 00 00”为 Data 区域的数据（将“C8 00 00 00”转换为十进制数为 200），第十二、十三字节“00 00”为保留区数据，最后两个字节“0D 0A”代表数据帧结束。

当单击“扣费”按钮对电子钱包进行扣费时，上位机向下位机发送的数据帧命令为：

```
EE CC 09 03 00 00 04 C8 00 00 00 00 00 0D 0A
```

数据帧解析如下。

头两个字节“EE CC”为帧头；第三字节“09”代表电子钱包扣费；第四字节“03”代表操作对象为 14443 高频模块；第五字节“00”代表卡选择模式为“不在选择模式”，第六、七字节“00 04”代表 Data 区域的数据长度为 4 个字节；第八字节开始的 4 个字节代表钱数，即“C8 00 00 00”为 Data 区域的数据（将“C8 00 00 00”转换为十进制数为 200）；第十二、十三字节“00 00”为保留区数据；最后两个字节“0D 0A”代表数据帧结束。

当单击“查询”按钮对电子钱包进行余额查询时，上位机向下位机发送的数据帧命令为：

```
EE CC 0A 03 00 00 01 00 00 00 0D 0A
```

数据帧解析如下。

头两个字节“EE CC”为帧头；第三字节“0A”代表电子钱包查询；第四字节“03”代表操作对象为 14443 高频模块；第五字节“00”代表卡选择模式为“不在选择模式”，第六、七字节“00 01”代表 Data 区域的数据长度为 1 个字节；第八字节“00”为 Data 区域的数据；第九、十字节“00 00”为保留区数据；最后两个字节“0D 0A”代表数据帧结束。

4.6.2 操作步骤

（1）打开上位机软件，启动 RFID 原理机的电源，用串口线建立通信连接。

（2）参照 4.1 节的操作步骤，读取卡片的 UID。将卡片放到识别区（如果连接了外接天线，将卡片放置在天线旁边），在执行寻卡操作之前需要将串口（默认波特率为 115200，若改成其他值，可能造成无法正确接收到数据）打开并发送“请求所有”操作请求，然后完成寻卡操作。

（3）参照 4.2 节的操作步骤完成认证操作。

（4）如图 4-43 所示，电子钱包功能区主要有三个输入框和三个按钮。第一个输入框为充值输入框，第二个输入框为扣费输入框，第三个输入框为余额输入框，输入框输入的数值不

能过大，否则会导致充值失败。三个按钮分别对应充值、扣费、查询功能，每次充值成功或者扣费成功，都会执行一次余额查询功能。

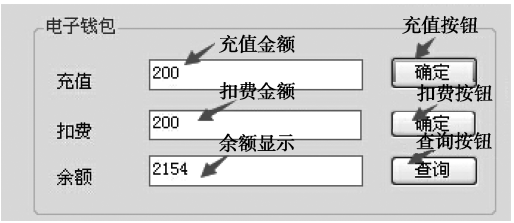


图 4-43　电子钱包

“充值”在充值过程中可能会由于一些原因（如卡片不在读卡区、输入金额过大等）会导致充值失败，充值失败返回如图 4-44 所示。充值成功后，程序将会对电子钱包余额进行查询，如图 4-45 所示。

图 4-44　充值失败

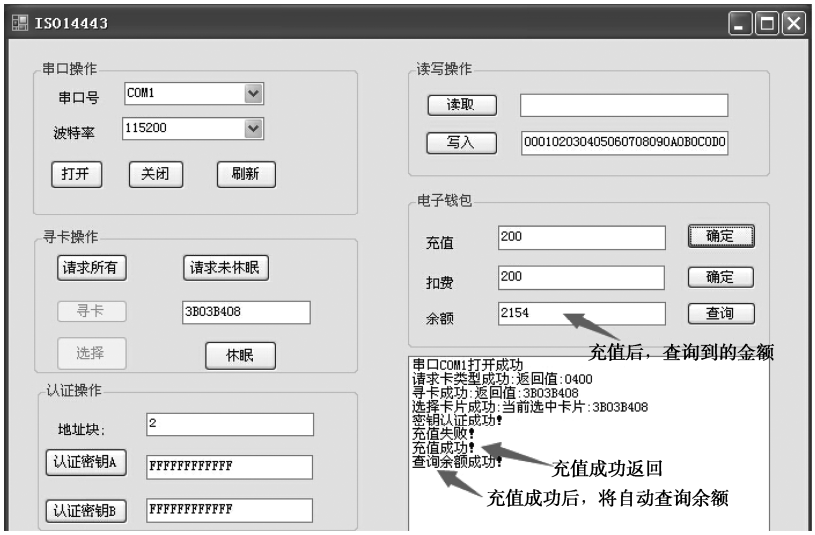


图 4-45　充值成功

“扣费”在扣费过程中可能会由于一些原因（如卡片不在读卡区、扣费金额大于余额等）会导致扣费失败。扣费成功后，程序将会对电子钱包余额进行查询，如图 4-46 所示。

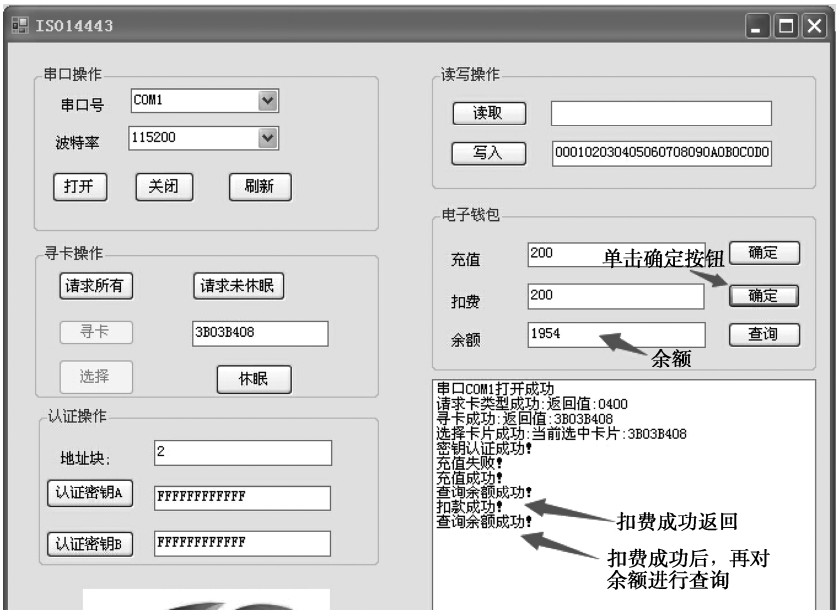


图 4-46　扣款操作

4.6.3　程序设计：电子钱包功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境下创建 C#窗体应用程序；编写代码，实现电子钱包功能。主要目的是了解 Microsoft Visual Studio 2010 开发环境及窗体应用程序建立；掌握程序通过串口与 RFID 实验系统平台建立连接并通信的原理和方法；了解 RFID 实验平台 COM 协议并实现电子钱包功能。

本设计所需设备如下。

（1）硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），ISO 14443（高频 13.56MHz）RFID 原理模块（基于 32 位 ARM STM32 嵌入式处理器），ISO 14443 卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计主要原理：上位机应用程序采用 C#语言开发，遵守 C#编程规范，电子钱包功能根据 RFID 高频模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 RFID 高频模块连接；上位机程序根据 RFID 高频模块 COM 协议通过串口与 RFID 高频模块进行通信，最终完成电子钱包的功能。

2．设计步骤

第一步：创建 C#窗体应用程序。

启动 Microsoft Visual Studio 2010 开发平台，创建一个如图 4-47 所示的 C#窗体应用程序，命名为“ISO14443”，创建的详细方法可参考第 3 章。

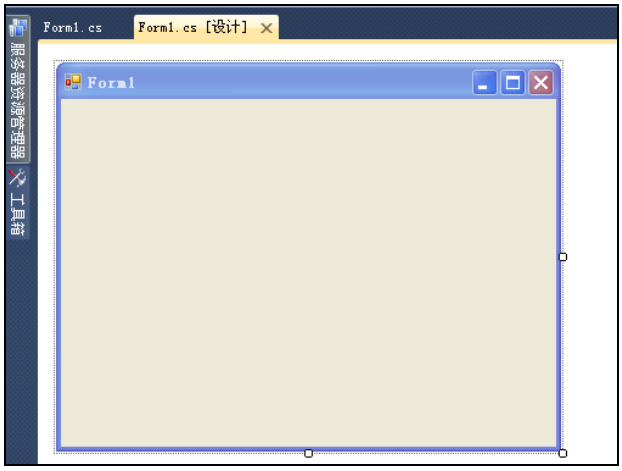


图 4-47　新建窗体

第二步：设计功能界面。

（1）选中新建的窗体，右击界面，在弹出的快捷菜单中选择“属性”选项，将 Text 属性改为“ISO14443”。

（2）打开工具箱 ，拖出一个 GroupBox，将 Text 属性改为“串口操作”，拖出两个 Label 放到串口操作的 GroupBox 中，并将其 Text 属性分别改为 “串口号”与“波特率”，拖出两个 ComboBox 放到串口操作的 GroupBox 中，并将 ComboBox 与“串口号”对应的 Name 属性改为“PortCmb”，将 ComboBox 与“波特率”对应的 Name 属性改为“BaudRateCmb”，并将 Items 属性设置为“9600、19200、57600、115200”，如图 4-48 所示。

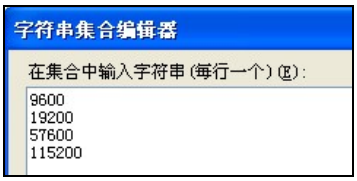


图 4-48　字符串集合编辑器

将 Text 属性设置为“115200”；拖出三个 Button 放到 GroupBox 中，分别将其 Text 属性设置为“打开”、“关闭”、“刷新”，Name 属性分别对应为“OpenBtn”、“CloseBtn”、“RefreshBtn”。

（3）再拖出一个 GroupBox，将 Text 属性改为“寻卡操作”，拖出 3 个 Button 放到寻卡操作的 GroupBox 中，将其 Text 属性分别设置为“请求所有”、“请求未休眠”、“寻卡”、“选择”，将对应的 Name 属性设置为“AllBtn”、“NoSleepBtn”、“FindBtn”、“SelectBtn”；拖出一个 TextBox 控件，将 Name 属性设置为“txt_Snr”。

（4）拖出一个 ListBox 控件，将其 Name 属性设置为“receive”，拖出一个 Button 放到其中，将 Name 属性设置为“receive”，Text 属性为“清空”。

（5）拖出一个 GroupBox，将 Text 属性改为“认证操作”，拖出一个 Label，将其 Text 属性改为“地址块”，拖出两个 Button，分别将 Text 属性设置为“认证密钥 A”、“认证密钥 B”，将对应的 Name 属性设置为“AuthABtn”、“AuthBBtn”，拖出 3 个 TextBox 分别将其 Name 属性设置为“txt_Address”、“txt_KeyA”、“txt_KeyB”。

（6）拖出一个 GroupBox，将 Text 属性改为“读写操作”，拖出两个 Button，将 Text 属性

分别设置为“读取”与“写入”，将 Name 属性设置为“ReadBtn”和“WriteBtn”，拖出两个 TextBox，将 Name 属性设置为“txt_DataOut”与“txt_DataIn”。再拖出一个 Button，将 Text 属性设置为“休眠”，Name 的属性为“SleepBtn”。

（7）拖出一个 GroupBox，将 Text 属性改为“电子钱包”，拖出三个 label，将 Text 属性分别设置为“充值”、“扣费”和“余额”，拖出三个 TextBox，将其 Name 属性分别设置为“txt_Recharge”、“txt_Pay”和“txt_Balance”，拖出三个 Button，将其 Text 属性分别设置为“确定”、“确定”和“查询”，对应的 Name 属性为“RechargeBtn”、“PayBtn”和“QueryBtn”。

完成后的功能界面如图 4-49 所示。

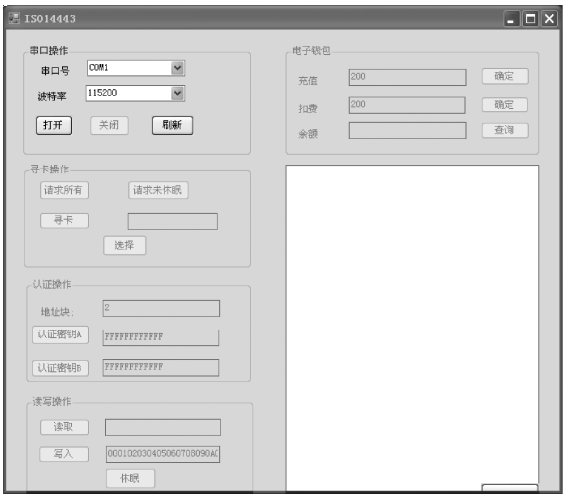


图 4-49　电子钱包界面设计

第三步：编写代码实现功能。

电子钱包程序执行流程图如图 4-50 所示。

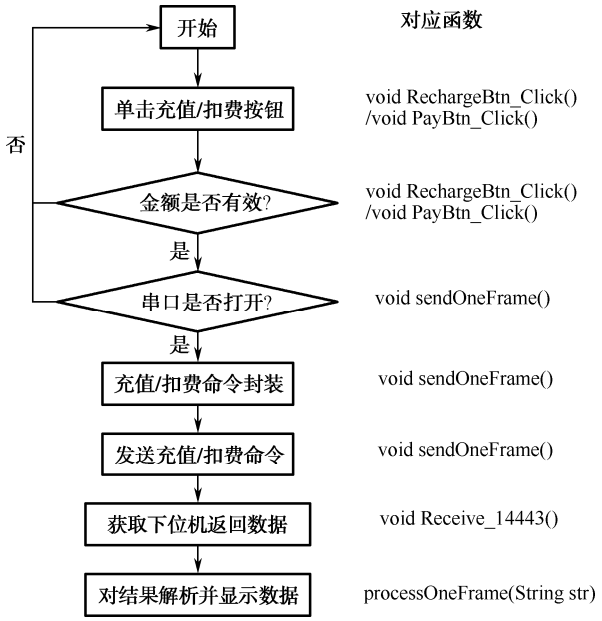


图 4-50　电子钱包程序执行流程图

部分主要代码解析如下：

```
//充值按钮事件
private void RechargeBtn_Click(object sender, EventArgs e)
{
    String RechargeNum = txt_Recharge.Text;          //字符串类型保存充值的金额
    int num;                                         //整型变量
    Int32.TryParse(RechargeNum.Trim(), out num);     //将金额数据类型转换
    if (RechargeNum.Equals("") || num == 0)          //判断输入的金额是否为空或为 0
    {
        ShowInfo("充值金额输入错误，请重新输入！");
        txt_Recharge.Focus();
        return;   //输入的金额不合法将跳出函数,不再继续往下执行
    }
    Byte[] data = new Byte[4];
    data[3] = (byte)((num >> 24) & (~(0x00)));       //充值金额转换成二进制数
    data[2] = (byte)((num >> 16) & (~(0xff00)));
    data[1] = (byte)((num >> 8) & (~(0xffff00)));
    data[0] = (byte)((num >> 0) & (~(0xffffff00)));
    sendOneFrame(0x08, data);     //发送一帧命令,0x08 标明该命令为充值命令
}
//扣费按钮事件
private void PayBtn_Click(object sender, EventArgs e)
{
    String payNum = txt_Pay.Text;                    //字符串类型保存扣费的金额
    int num; //整型变量
    Int32.TryParse(payNum.Trim(), out num);          //将金额数据类型转换
    if (payNum.Equals("") || num == 0)               //判断输入的金额是否为空或为 0
    {
        ShowInfo("充值金额输入错误，请重新输入！");
        txt_Pay.Focus();
        return;     //输入的金额不合法将跳出函数,不再继续往下执行
    }
    Byte[] data = new Byte[4];
    data[3] = (byte)((num >> 24) & (~(0x00)));       //充值金额转换成二进制数
    data[2] = (byte)((num >> 16) & (~(0xff00)));
    data[1] = (byte)((num >> 8) & (~(0xffff00)));
    data[0] = (byte)((num >> 0) & (~(0xffffff00)));
    sendOneFrame(0x09, data);     //发送一帧命令,0x09 标明该命令为扣费命令
}
//查询按钮事件
private void QueryBtn_Click(object sender, EventArgs e)
{
    Byte[] a = new Byte[1];                          //Data 区域数据
    a[0] = 0x00;                                     //为 Data 区域数据赋值
    sendOneFrame(0x0A, a);                           //发送一帧命令,0x0A 标明此命令为余额查询命令
}
/*其他函数及功能
```

```
        其他函数声明及功能已在本章其他设计部分中说明(具体函数定义见源码)
        */
```

第四步：编译生成程序运行。

（1）在菜单中选择“调试”→“启动调试”选项或按 F5 键生成程序并运行。

（2）选择正确的波特率和串口号，单击“打开”按钮，将 14443 的电子标签放置在读卡区，串口打开成功后单击“寻卡”按钮，执行相应操作，结果如图 4-51 所示。

图 4-51　电子钱包功能实现

第 5 章　RFID 超高频 ISO 18000-6 协议原理及实践开发

本章所述内容所使用设备包括桂林华智 RFID 物联网教学科研平台实验箱 ISO 18000-6 协议读写器、ISO 18000-6 卡片、串口线。主要操作内容：通过使用上位机软件，与 ISO 18000-6 模块下位机进行通信，完成对 ISO 18000-6 标签的操作。通过本章的学习，掌握对 ISO/IEC 18000-6 协议读写器的操作，了解相关协议及工作原理。

（1）读写 ISO 18000-6 协议标签的读写器工作电压 5V，工作温度 0℃～60℃，工作频段是 UHF 900MHz，这种读写器广泛应用于物流、车辆门禁、人员门禁、生产流水线等领域，与汽车衡、轨道衡等配套能够实现物流的管理，能够同时读写多个标签。

（2）产品电路图，如图 5-1 所示。

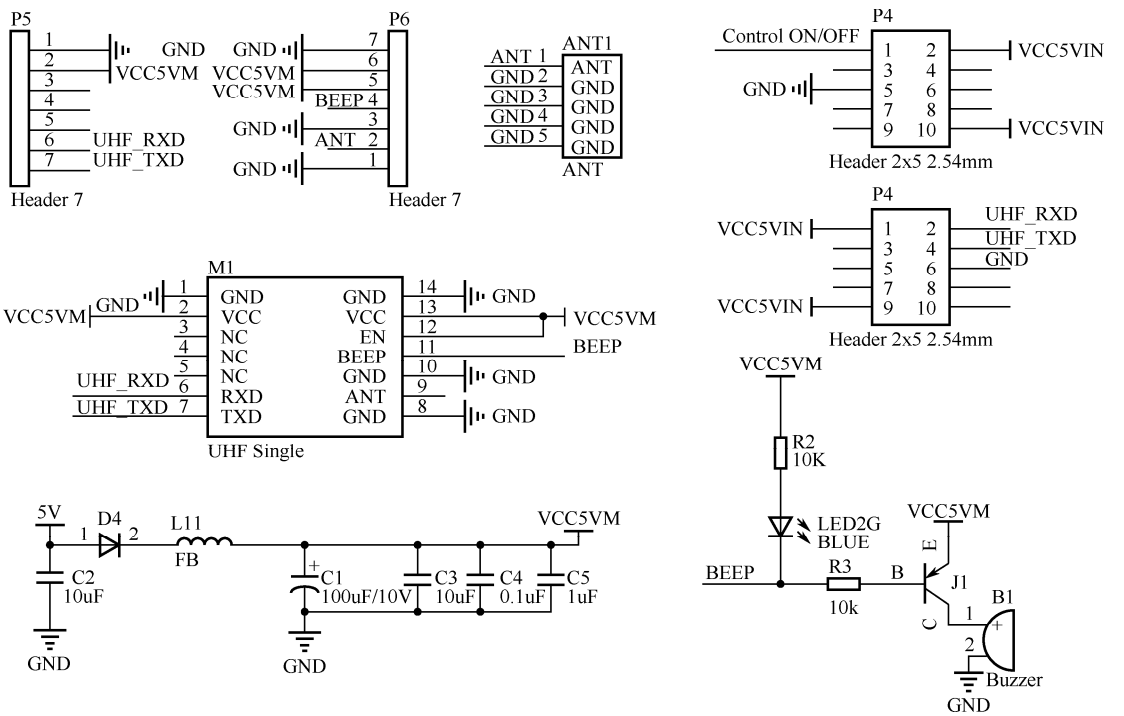


图 5-1　产品电路图

（3）产品平面图，如图 5-2 所示。

（4）工作原理简介。

JR2010 模块是超小型化的 UHF RFID 读写器核心部件，里面集成了 PLL、发射、接收、耦合器及 MCU（微控制器）等部件，如图 5-3 所示。模块内置电源管理，3.3V～6.5V 的超宽电压试用范围，最大程度简化用户的二次开发，可以很方便地通过 API 函数库控制模块工作。

JR2010 工作频率为 840MHz～930MHz，支持协议有 EPC C1 GEN2/ISO 18000-6C，低电压工作 3.3V～6.5V，接口 UART。

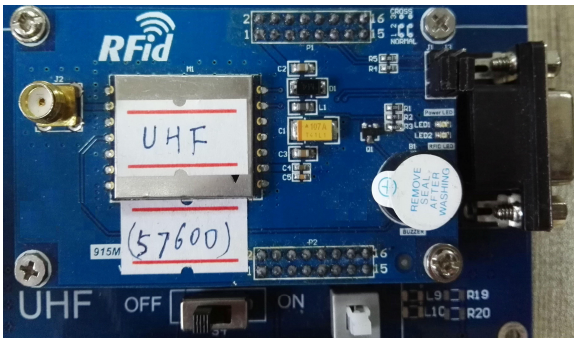


图 5-2　产品平面图

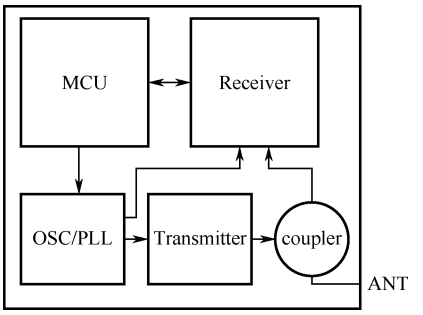


图 5-3　工作原理图

5.1　UHF 900MHz ISO 18000-6 识别标签号

5.1.1　协议原理

1．ISO/IEC 18000-6 国际标准协议第六部分协议要求

ISO/IEC18000-6《信息技术——针对物品管理的射频识别（RFID）第六部分：针对频率为 860MHz～930MHz 无接触通信空气接口参数》的阅读器与应答器之间的物理接口、协议和命令以及防冲突判断机制。

Type A 协议的通信机制是基于一种“阅读器先发言”的，即基于阅读器的命令与应答器的回答之间交替发送的机制。整个通信中的数据信号定义为“0”、“1”、“SOF”和“EOF”4 种。

通信中的数据信号的编码和调制方法定义如下。

（1）阅读器到应答器之间的通信传输：阅读器发送的数据采用 ASK（调制载波幅度）进行调制，调制深度是 30%（误差不超过 3%）；数据编码采用脉冲宽度编码（PIE）来编码数据，即通过定义下降沿之间的不同宽度来表示不同数据信号。

（2）应答器到阅读器之间的传输连接：应答器通过反向散射给阅读器来传输信息；数据编码采用 FMO 编码，数据速率是 40kbps。

（3）防冲突采用时隙 ALOHA 算法。

2．数据命令解析

上位机向下位机发送命令格式如下：

SOF	LEN	CMD	PAYLOAD	*CRC16	EOF
1	EBV	1	-	2	1

下位机对上位机响应命令格式如下：

SOF	LEN	CMD	Status	PAYLOAD	*CRC16	EOF
1	EBV	1	1	-	2	1

数据命令解析如下（长度的单位均为字节）。

SOF：0xAA。

LEN：SOF～EOF 之间数据长度，即 LEN+CMD+PAYLOAD+CRC16。

CMD 位说明如下：

7	6～0
0：数据包没有 CRC 校验 1：数据包为 CRC 校验	模块命令

Status：发送命令中没有 Status 字段，只存在响应数据包中。高 4 位是通用响应标志，低 4 位是各命令特有的响应标志。

Status 位说明如下：

7	6	5	4	3～0
0：执行命令失败 1：执行命令成功	1：CRC 校验失败 0：CRC 校验成功	保留	保留	各命令状态见命令表

PAYLOAD：各命令需要传送的实际数据，数据长度在各命令格式中定义，最长 512 字节。

CRC16：对 LEN CMD PAYLOAD 部分计算的 CRC16 值，当上位机的命令的 CRC16 校验失败时，返回固定格式的响应，且 Status 值为 0xC0。

EOF：0x55 标识一帧结束。

3．具体发送的命令

本节内容主要为寻卡操作与设计，寻单卡操作上位机向下位机发送的数据命令为“AA 02 10 55”（02 代表“02 10”共两个字节，10 代表识别单标签命令）。

寻多卡操作上位机向下位机发送的数据命令为“AA 03 11 03 55”（03 代表“02 10 03”共三个字节，11 代表识别多标签命令）。

5.1.2 操作步骤

（1）如果连接设备使用的是 USB 口转串口线，先安装 USB 转 UART 驱动。

（2）打开 RFID 原理机的电源，建立通信连接，将计算机设备与超高频 UHF 900MHz 读写器模块通过串口线（或 USB 口转串口线）连接，打开 UHF 实验软件，进行实验。

实验软件界面如图 5-4 所示。

① 如果没有连接上串口，打开串口就会失败。

② 波特率默认为 57600，如果选择其他波特率，就可能会打开串口失败。

③ 如果启动实验软件后，在串口栏未显示串口号，请检查串口连接是否正常，再单击“刷新”按钮，对串口进行刷新操作，获取到串口号。

④ 关闭串口后，串口将被关闭，所有操作、指令都将无法正常执行。

（3）打开与关闭串口操作。

在正确建立通信后，打开/关闭串口成功返回如图 5-5 所示的信息。

（4）识别标签号操作。

在进行识别标签号操作之前，必须将串口打开。在串口打开成功后才能进行识别标签号操作，否则会提示出错。

图 5-4　软件界面

图 5-5　打开与关闭串口

将卡片放到识别区（如果连接了外接天线，则将卡片放置在天线旁边。）单击“寻卡”按钮，执行寻卡操作。寻卡操作过程如图 5-6 所示。

图 5-6　寻卡操作过程

寻卡成功返回如图 5-7 所示的信息。

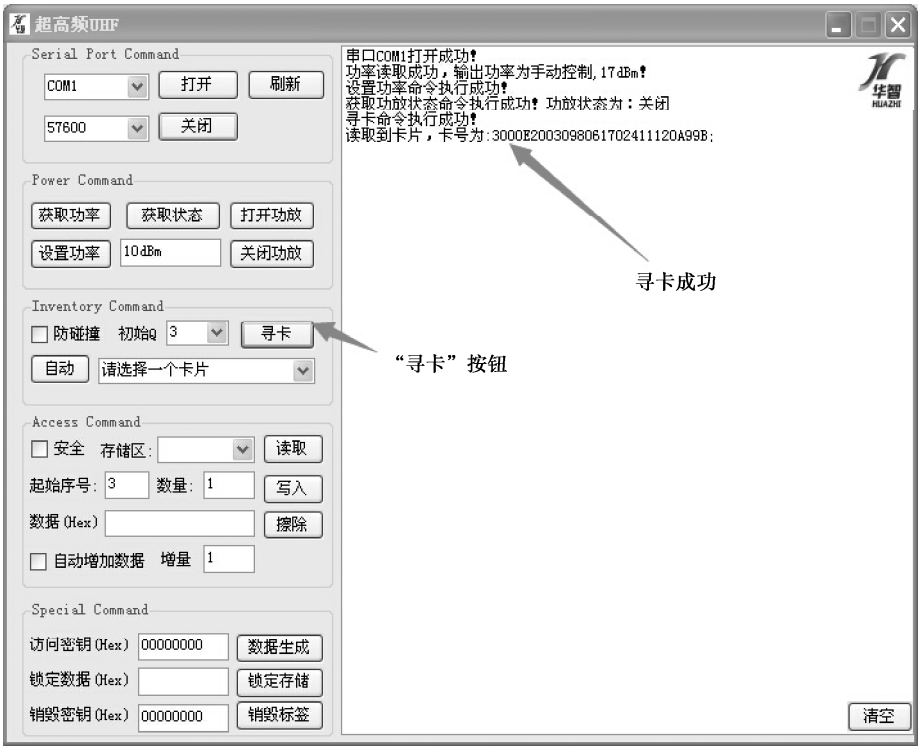


图 5-7　寻卡操作

（5）识别多标签号操作。

当“防碰撞”复选框被选中时，“识别标签”操作为“防碰撞识别”操作，启动标签识别循环和启动防碰撞功能，对多张标签进行识别时使用该命令。发送命令时需指定防碰撞识别的“初始 Q”值。“初始 Q”值范围为 0～15，使用默认“初始 Q”值为 3（也可更改为其他适当的值）。该命令的响应方式与单标签循环识别命令一致（射频识别系统中 UHF 阶段的 Q 值防碰撞算法，利用参数 Q 值的变化动态地改变识别帧中的时隙数，以获得更高的识别效率）。

执行寻多卡前需要选中“防碰撞”复选框，寻多卡成功返回如图 5-8 所示的信息。

如果未选中“防碰撞”复选框，将多个标签放入寻卡区进行寻卡操作，则可能导致识别标签失败。识别标签失败返回如图 5-9 所示的信息。

图 5-8　寻多卡成功

图 5-9　寻卡失败

（6）自动识别标签操作。

自动执行识别标签操作，就是让读写器不断地进行标签识别，读写器将识别到的标签号

返回给上位机程序。在进行自动识别标签时建议选中“防碰撞”复选框，因为在自动识别标签时有可能会有多个标签在读写区中，如果未选中，则在寻卡区有多个标签存在，将可能导致寻卡失败。自动寻卡命令如图 5-10 所示。

图 5-10　自动寻卡

单击“自动”按钮后，按钮将变成“停止”按钮，再次单击将停止自动识别标签操作，自动识别标签操作成功返回如图 5-11 所示的信息。

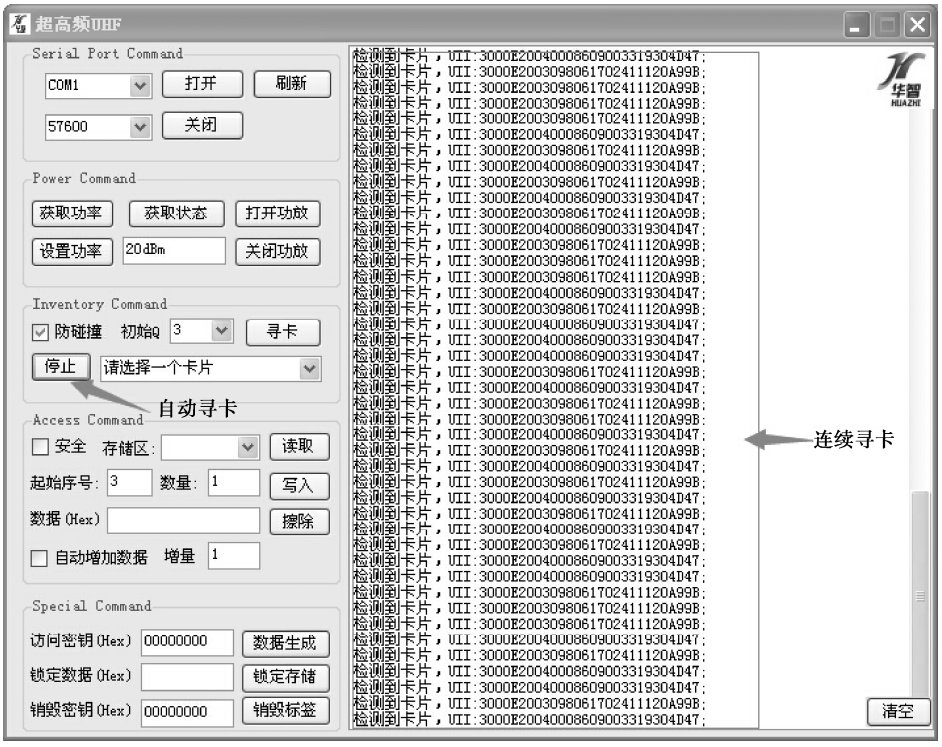


图 5-11　自动寻卡操作

5.1.3　程序设计：识别标签号功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境下创建 C#窗体应用程序；编写代码，实现识别标签号功能。主要目的是了解 Microsoft Visual Studio 2010 开发环境及窗体应用程序建立；掌握程序通过串口与 RFID 实验系统平台建立连接并通信的原理和方法；了解 RFID 实验平台 COM 协议并实现识别标签号功能。

本设计所需设备如下。

（1）硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），

ISO 18000-6（超高频 900MHz）RFID 原理模块（基于 32 位 ARM STM32 嵌入式处理器），ISO 18000-6 卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计主要原理：上位机应用程序采用 C#语言开发，遵守 C#编程规范，识别标签号功能根据 RFID 超高频模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 RFID 超高频模块连接；上位机程序根据 RFID 超高频模块 COM 协议通过串口与 RFID 超高频模块进行通信，最终完成识别标签号的功能。

2．设计步骤

第一步：创建 C#窗体应用程序。

启动 Microsoft Visual Studio 2010 开发平台，创建一个如图 5-12 所示的 C# Windows 窗体应用程序，命名为“UHF ISO 18000-6”。

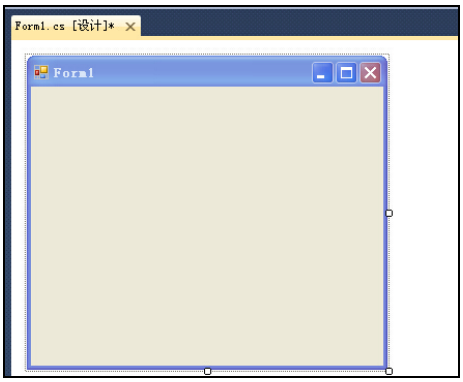


图 5-12　新建窗体

第二步：设计功能界面。

（1）选中新建的窗体，右击界面，在弹出的快捷菜单中选择“属性”选项，将 Text 属性改为“UHF ISO 18000-6”。

（2）打开工具箱  ，拖出一个 GroupBox，将 Text 属性改为“串口操作”；拖出两个 ComboBox 放到串口操作的 GroupBox 中，将其 Text 属性分别改为“串口号”与“波特率”，并将 ComboBox 与“串口号”对应的 Name 属性改为“cmb_UHF_PortName”；将 ComboBox 与“波特率”对应的 Name 属性改为“cmb_UHF_BaudRate”，并将 Items 属性设置为 9600、19200、57600、115200，如图 5-13 所示。

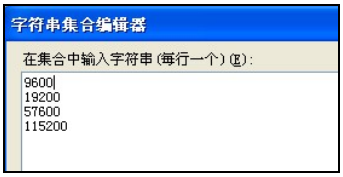


图 5-13　字符串集合编辑器

拖出三个 Button 放到 GroupBox 中，分别将其 Text 属性设置为“打开”、“关闭”、“刷新”，Name 属性分别对应为“btn_UHF_Open”、“btn_UHF_Close”、“btn_UHF_Refresh”。

（3）拖出一个 GroupBox，将其 Text 属性改为“寻卡命令”；拖出一个 CheckBox 放在寻卡

操作的 GroupBox 中，将 Text 属性设置为“防碰撞”及 Name 属性设置为“chk_UHF_Anticoll”；再拖入一个 Label，将其 Text 属性设为“初始 Q”；拖入一个 ComboBox，将 Name 属性设置为“cmb_UHF_IntQ”；拖入两个 Button，将 Text 属性分别设置为“寻卡”和“自动寻卡”，Name 属性分别设置为“btn_UHF_Inv”和“btn_UHF_Inventory”；最后拖入一个 ComboBox，将 Name 属性设置为“cmb_UHF_UII”，Text 属性设置为“请选择一个标签”。

（4）在窗体的右方拖入一个 ListBox，将 Name 属性设置为“lst_UHF_Info”，并在其右下方拖入一个 Button，将 Text 属性设置为“清空”及 Name 属性设为“btn_UHFlist_Clear”。

最后设计好的界面如图 5-14 所示。

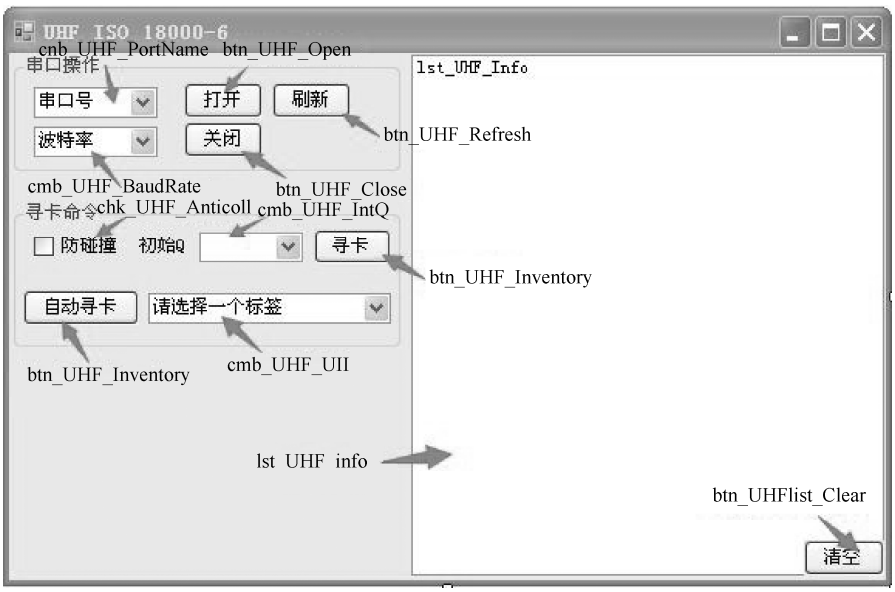


图 5-14　寻卡功能界面设计

第三步：加入动态链接库资源。

找到新建工程的路径，将 M900_API.dll 文件复制到 UHF ISO 18000-6\UHF ISO 18000-6\bin\Debug 目录下，如图 5-15 所示。

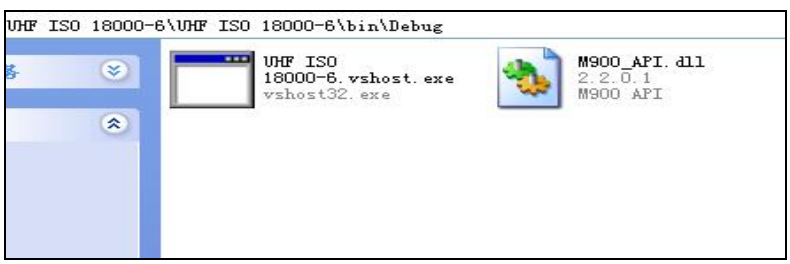


图 5-15　加入动态链接库资源

第四步：编写代码实现功能（此处只附主要代码）。

UHF 寻卡功能执行过程如图 5-16 所示。

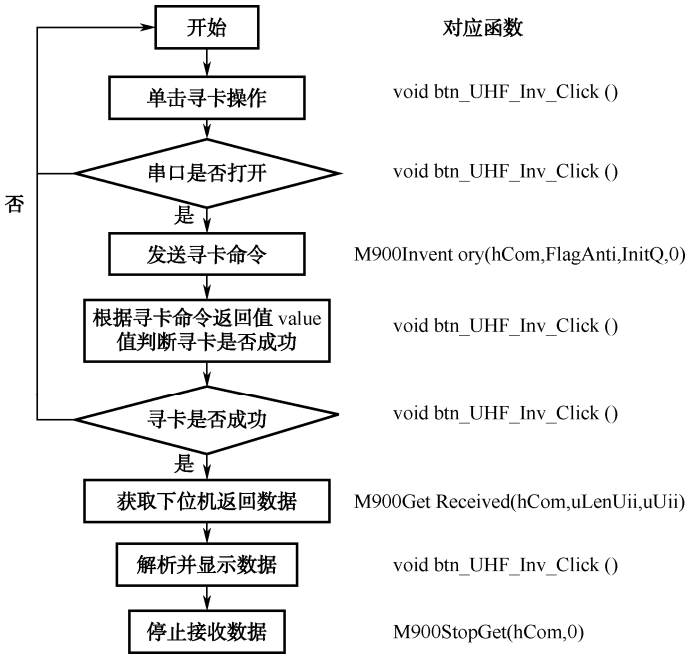


图 5-16　UHF 寻卡功能执行过程

部分代码解析如下：

```
//寻卡按钮事件
private void btn_UHF_Inv_Click(object sender, EventArgs e)
{
    if (!UHF_IsOpen)              //判断串口是否已经打开
    {
        AddUHFList("错误：串口尚未打开，不能操作！");
        return;                   //若串口未打开，则跳出，不再往下执行
    }
    if ((chk_UHF_Anticoll.Checked) && (cmb_UHF_IntQ.SelectedIndex < 0))
    {
        cmb_UHF_IntQ.SelectedIndex = 3;//若选中“防碰撞”复选框，则选择默认“初始 Q”为 3
    }
    Byte FlagAnti = (Byte)((chk_UHF_Anticoll.Checked) ? 0x01 : 0x00);
    //若选中“防碰撞”复选框，则为字节型变量 FlagAnti 赋值 0x01，否则赋值 0x00
    Byte InitQ=(Byte)((chk_UHF_Anticoll.Checked)?cmb_UHF_IntQ.SelectedIndex : 0x00);
    //初始 Q
    Int32 value = M900Inventory(hCom, FlagAnti, InitQ, 0);
    //函数功能：该函数启动 M900 的识别循环
    //函数参数：HANDLE hCom：通信端口句柄
    //UCHAR flagAnti：是否使用防碰撞识别功能。1：防碰撞识别；0：单标签识别
    //UCHAR initQ：防碰撞识别过程的初始 Q 值，flagAnti 为 1 时有效
    //UCHAR flagCrc：1：使用 CRC 功能；0：不使用 CRC 功能
    //返回值：1：成功启动 M900 的识别循环
    //其他：启动 M900 的识别循环失败
    if (value != 1) //判断函数返回是否等于 1，如果不等于 1 则寻卡命令执行失败
    {
```

```
        AddUHFList(String.Format("寻卡命令执行失败！"));
        return;
    }
    AddUHFList(String.Format("寻卡命令执行成功！"));
    UHF_TagUIIs = new ArrayList();  //新建集合
    Byte[] uLenUii = new Byte[1];      //新建一个长度的字节数组，用于保存标签长度
    Byte[] uUii = new Byte[255];       //标签 UII
    String strUii; //字符串，用于提取标签号
    value = M900GetReceived(hCom, uLenUii, uUii); //该函数读取 M900 返回的标签 UII
    //函数原型
    //int WINAPI M900GetReceived (HANDLE hCom, UCHAR* uLenUii, UCHAR* uUii);
    //返回值 value：1：成功读取标签的 UII；其他：读取标签的 UII 失败
    //输入参数
    //'HANDLE hCom：通信端口句柄。
    //'UCHAR* uLenUii：标签 UII 的长度，1 个字节。
    //'UCHAR* uData：标签 UII，至少 66 个字节。
    while (value == 1)                    //当 alue 值为 1 时进入循环，获取标签号并显示
    {
        strUii = "";
        Int32 length = uLenUii[0];     //获取标签 UII 的长度
        for (Int32 i = 0; i < length; i++)
        {
            strUii += String.Format("{0:X2}", uUii[i]);       //取得标签号
        }
        if (UHF_TagUIIs.IndexOf((object)strUii) < 0)
        {
            UHF_TagUIIs.Add(strUii);
            AddUHFList(String.Format("读取到卡片，卡号为:{0};", strUii)); //显示卡号
            value = M900GetReceived(hCom, uLenUii, uUii);     //再次获取 value 值
        }
        else
        {
            value = 0;
        }
    }
    cmb_UHF_UII.Items.Clear();                    //清空标签号显示下拉框
    if (UHF_TagUIIs.Count > 0)
    {
        foreach (String struii in UHF_TagUIIs)
        {
            cmb_UHF_UII.Items.Add(struii);    //在标签显示下拉框显示标签号
        }
        cmb_UHF_UII.Text = "请选择一个卡片";
    }
    else
    {
        AddUHFList("接收数据命令执行失败，没有读取到有效的卡片。");
        AddUHFList("请将电子标签置于天线辐射场内！");
```

```
        }
        value = M900StopGet(hCom, 0);                    //停止 M900 的识别循环
        //返回值 1：成功停止识别。其他：停止识别失败
        //输入参数 HANDLE hCom：通信端口句柄
        //UCHAR flagCrc：是否使用 CRC16 验证功能，1：使用 CRC 功能；0：不使用 CRC 功能
    }
```

第五步：编译生成程序运行。

单次寻卡操作过程如图 5-17 所示。

图 5-17　单次寻卡

自动寻卡操作过程如图 5-18 所示。

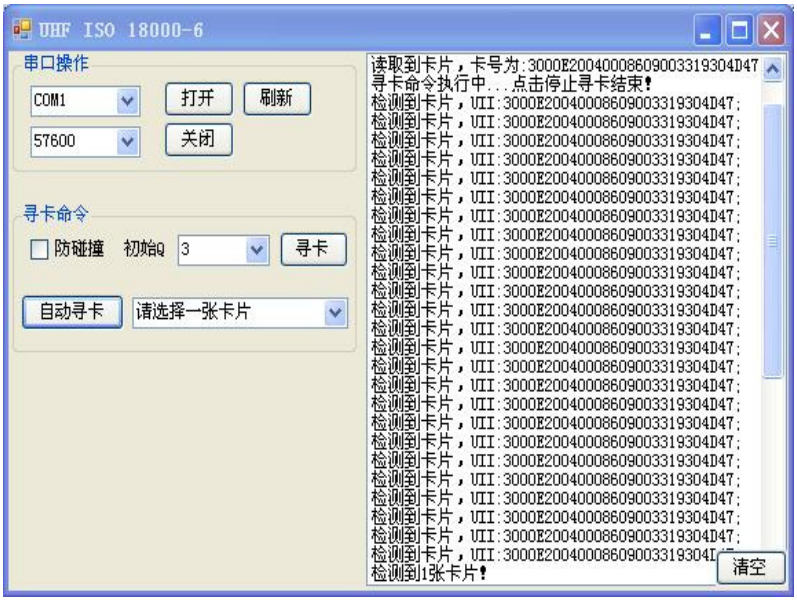


图 5-18　自动寻卡

5.2 UHF 900MHz ISO 18000-6 功率设置操作

5.2.1 协议原理

（1）ISO/IEC 18000-6 国际标准协议第六部分协议要求。

（2）具体的数据帧格式参照 5.1.1 协议原理部分。

（3）具体发送的命令。

本节内容主要为功率设置的操作与设计。

完成获取状态操作：上位机向下位机发送的数据命令为“AA 02 00 55”（02 代表 0x02 0x00 共两个字节，0x00 代表状态获取操作）。

完成功率获取操作：上位机向下位机发送的数据命令为“AA 02 01 55”（02 代表 0x02 0x01 共两个字节，0x01 代表功率获取操作）。

完成功率设置操作：上位机向下位机发送的数据命令为“AA 04 02 03 14 55”（04 代表 04 02 03 14 共四个字节，将第五字节的 0x14 转换成十进制数为 20，即为要设置的功率 20dBm）。

5.2.2 操作步骤

（1）按照 5.1 节的步骤，将串口连接成功，打开上位机实验软件，选择对应的串口号，选择默认的波特率“57600”，打开串口。

图 5-19　功率获取与设置区

（2）查看功率获取与设置区，如图 5-19 所示。

① 获取状态：UHF 的设置状态，包括功率的大小及功放的打开状态。

② 获取功率：读取天线发射的功率大小。

③ 设置功率：设置工作功率。

④ 打开功放和关闭功放：需要设置功率，才能进行打开和关闭功放的操作，且不能执行其他操作。

（3）获取功率：读取天线发射的功率大小，获取功率成功返回如图 5-20 所示的信息。

图 5-20　获取功率

（4）设置功率，设置功率即可以修改发射功率，设置功率成功返回如图 5-21 所示的信息。功率设置的范围为 5dBm～20dBm。相对来说，功率越高，读卡距离越远，当输出功率达到 20dBm，读卡距离达到最远。虽然可以修改代码使功率更高，但是不建议这样做。

图 5-21　设置功率

（5）获取状态：获取 UHF 的设置状态，包括功率的大小及功放的打开状态，获取设备状态成功返回如图 5-22 所示的信息。

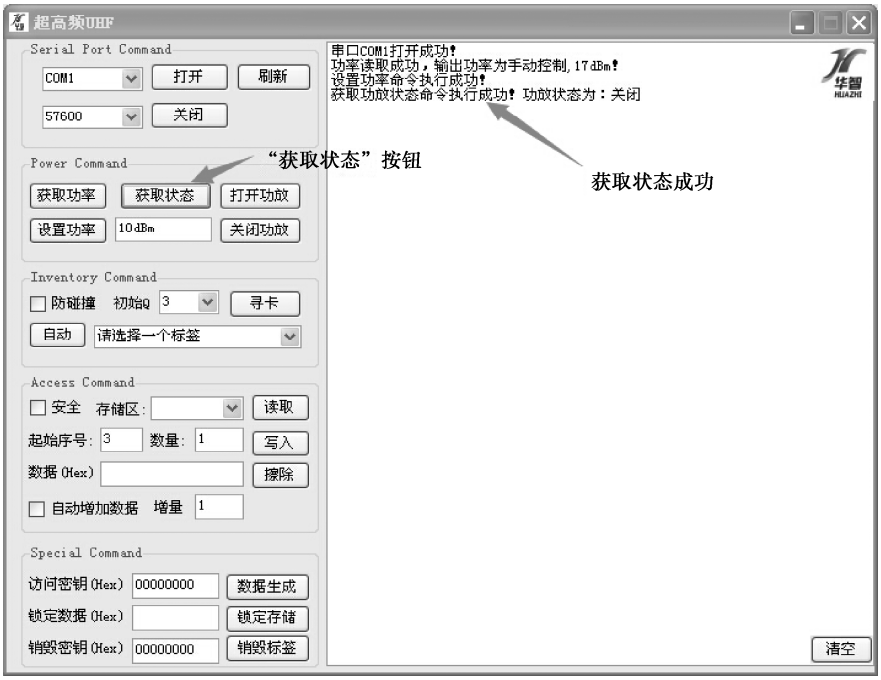


图 5-22　获取状态

5.2.3 程序设计：功率设置功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境下创建 C#窗体应用程序；编写代码，实现功率设置功能。主要目的是了解 Microsoft Visual Studio 2010 开发环境及窗体应用程序建立；掌握程序通过串口与 RFID 实验系统平台建立连接并通信的原理和方法；了解 RFID 实验平台 COM 协议并实现功率设置功能。

本设计所需设备如下。

（1）硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），ISO 18000-6（超高频 900MHz）RFID 原理模块（基于 32 位 ARM STM32 嵌入式处理器），ISO 18000-6 卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计主要原理：上位机应用程序采用 C#语言开发，遵守 C#编程规范，功率设置功能根据 RFID 超高频模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 RFID 超高频模块连接；上位机程序根据 RFID 超高频模块 COM 协议通过串口与 RFID 超高频模块进行通信，最终完成功率设置的功能。

2．设计步骤

第一步：在 5.1.3 节的基础上进行设计。

在原有界面上拖入一个 GroupBox，将 Text 属性设为“功率操作”；在“功率操作”的 GroupBox 中拖入 3 个 Button，将 Text 属性分别设置为“获取功率”、“获取状态”、“设置功率”，再将 Name 属性分别设置为“btn_UHF_GetPower”、“ btn_UHF_GetPaStatus”、“btn_UHF_SetPower”；最后拖入一个 TextBox，将 Name 属性设置为“txt_UHF_Power”。

完成上述步骤后，得到的界面如图 5-23 所示。

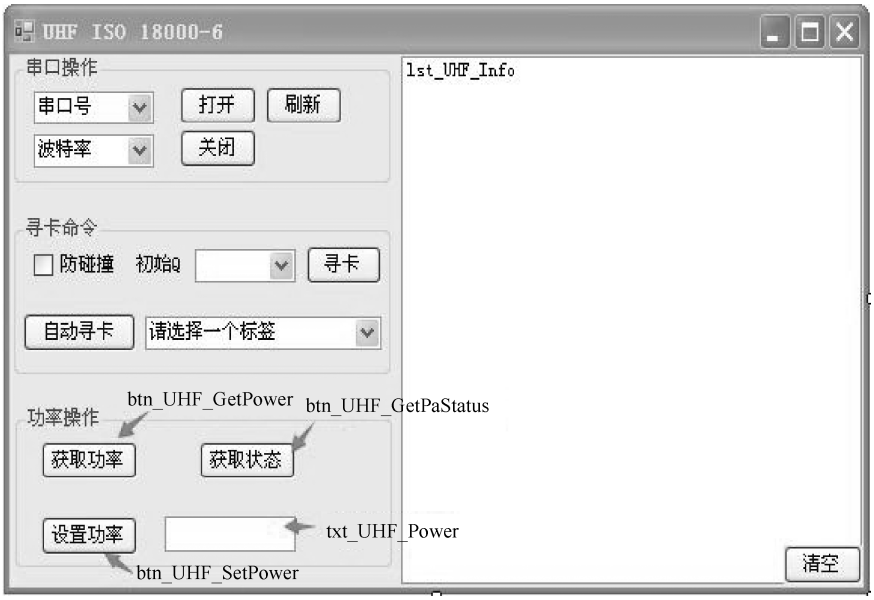


图 5-23　功率操作界面设计

第二步：编写代码实现功能（此处只附部分主要代码）。

UHF 功率设置功能执行过程如图 5-24 所示。

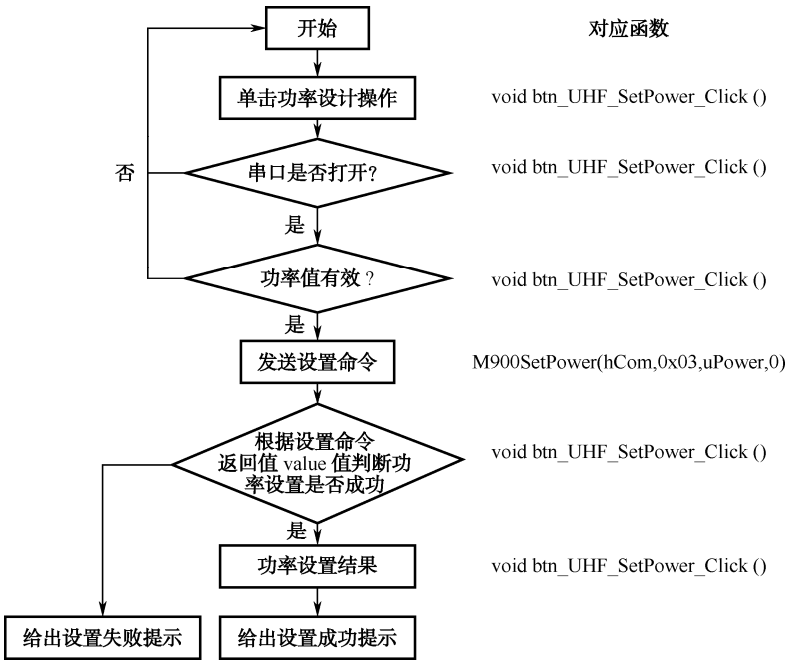


图 5-24　UHF 功率设置功能执行过程

部分代码解析如下：

```
//获取功率事件
private void btn_UHF_GetPower_Click(object sender, EventArgs e)
{
    if (!UHF_IsOpen)                    //判断串口是否打开
    {   AddUHFList("错误：串口尚未打开，不能操作！");
        return;
    }
    Byte[] uPower = new Byte[1];        //一个长度的字节数组
    Int32 value = M900GetPower(hCom, uPower, 0);//函数功能为读取 M900 的功率设置
    //返回值 1：读取 M900 的功率设置成功。其他：读取 M900 的功率设置失败
    //输入参数 HANDLE hCom：通信端口句柄。UCHAR* uPower：M900 返回的 POWER 字节
    //UCHAR   flagCrc 1：使用 CRC 功能；0：不使用 CRC 功能
    if (value == 1)                     //获取功率成功
    {
        String ControlMode, Power;  //控制模式，功率值
        if (uPower[0] >= 0x80)          //判断 M900 返回的 Power 字节状态，自动控制
        {
            ControlMode = "自动控制";
            Power = ((Byte)(uPower[0] - 0x80)).ToString();
        }
        else                            //手动控制
        {
```

```
                ControlMode = "手动控制";
                Power = uPower[0].ToString();
            }
            AddUHFList(String.Format("功率读取成功，输出功率为{0},{1}dBm！", ControlMode,
                    Power));
            txt_UHF_Power.Text = Power + "dBm"; //显示功率值
        }
        else
        {
            AddUHFList(String.Format("获取功率命令执行失败！"));
            txt_UHF_Power.Text = "读取失败！";
        }
    }
    //设置功率事件
    private void btn_UHF_SetPower_Click(object sender, EventArgs e)
    {
        if (!UHF_IsOpen)                                //判断串口是否打开
        {
            AddUHFList("错误：串口尚未打开，不能操作！");
            return;
        }
        String strPower = txt_UHF_Power.Text.Trim();//字符串保存设置的功率值
        if (strPower.IndexOf("dBm") >= 0)               //输入的功率有效
            strPower = strPower.Remove(strPower.IndexOf("dBm"), 3);//去掉 dBm
        Byte uPower;
        try
        {
            uPower = Byte.Parse(strPower);              //将设置功率的数据值类型转换并保存
        }
        catch                                           //转换过程出现异常
        {
            AddUHFList("错误：您设置的功率值有误，请填写 5～20 之间的整数！");
            txt_UHF_Power.SelectAll();
            txt_UHF_Power.Focus();
            return;
        }
        if ((uPower < 5) || (uPower > 20))
        {
            AddUHFList("错误：您设置的功率值有误，请填写 5～20 之间的整数！");
            txt_UHF_Power.SelectAll();
            txt_UHF_Power.Focus();
            return;
        }
        Int32 value=M900SetPower(hCom, 0x03, uPower, 0);//函数功能：设置 M900 的功率
       //返回值 1：成功设置 M900 的功率设置。其他：设置 M900 的功率失败
       /*输入参数  HANDLE hCom：通信端口句柄
        UCHAR uOption：设置功率命令的 OPTION 字节
        UCHAR uPower：设置功率命令的 POWER 字节
```

```
            UCHAR   flagCrc：1：使用 CRC 功能；0：不使用 CRC 功能*/
            if (value == 1)    //设置功率设置成功
                AddUHFList(String.Format("设置功率命令执行成功！"));
            else               //设置失败
                AddUHFList(String.Format("设置功率命令执行失败！"));
        }
```

第三步：编译生成程序运行。

将 UHF 模块通过 USB 数据线与 PC 进行连接。编译运行程序，得到的运行结果如图 5-25 所示。

图 5-25　功率操作实现

5.3　UHF 900MHz ISO 18000-6 读取数据

5.3.1　协议原理

（1）ISO/IEC 18000-6 国际标准协议第六部分协议要求。

（2）读取标签数据（指定 UII）。

功能简介：该命令从指定的 UII 的目标标签中读取数据。

读取电子标签指定数据块的信息，如标签号、00:Rsv、01:UII、10:TID、11:USER 的数据信息。该功能操作有两种工作模式，分别是“非安全”和“安全”工作模式。

① 当“安全”复选框未被选中时，该操作工作在非安全工作模式下，此时用户读取电子标签的指定信息，将正常地进行读取标签存储区，不需要其他的验证。

② 当“安全”复选框被选中时，该操作工作在安全工作模式下，此时用户读取电子标签的信息，将需要密码验证，默认密码为 0x00000000。

（3）标签数据存取命令集。

发送数据命令格式如下：

数据段	SOF	LEN	CMD	APWD	BANK	PTR	CNT	UII	CRC16	EOF
长度	1	EBV	1	4	1	EBV	1		2	1

数据段说明：

① APWD：标签的数据访问密码。

当用户读取的数据存储区为非 Reserved 存储区时，APWD 将置为 0x00000000。

当用户读取的数据存储区为 Reserved 存储区时，APWD 为标签的访问密钥。

② BANK：标签的存储分区。

Reserved 存储区，BANK=0x00

UII 存储区，BANK=0x01

TID 存储区，BANK=0x02

User 存储区，BANK=0x03

③ PTR：标签存储区的起始地址，EBV 格式。

④ CNT：数据长度，以 WORD（2 字节）为单位，支持 CNT=0。

（4）具体发送的数据命令。

读取数据：（卡号为“3000E200400086090033193O4D47”，存储区为“00:Rsv”区，选中“安全”复选框，起始序号为 3，数量为 1），发送命令：

AA 17 13 00 00 00 00 03 01 30 00 E2 00 40 00 86 09 00 33 19 30 4D 47 55

发送数据命令解析：第一字节“0xAA”为数据帧开始标识，最后一个字节“0x55”为数据帧结束标识；第二字节“0x17”（转换成十进制数为 23）为从数据帧开始标识 0xAA 到数据帧结束标识 0x55 之间共有 23 个字节数据；第三字节“0x13”为读取数据命令；第四到第七字节“0x00 0x00 0x00 0x00”为标签的数据访问密码；第八字节“0x00”为要读取的存储区为 Rsv 区（UII 存储区：0x01；TID 存储区：0x02；User 存储区：0x03）；第九字节“0x03”为标签存储区的起始地址；第十字节“0x01”为读取数据的数量为 1×2 字节；第十一字节到第二十三字节为要进行数据读取的标签号。

返回命令：

AA 05 13 00 00 00 55（读到的数据为：0000）

返回数据命令解析：第一字节“0xAA”为数据帧开始标识，最后一个字节“0x55”为数据帧结束标识；第二字节“0x05”为从数据帧开始标识 0xAA 到数据帧结束标识 0x55 之间共有 5 个字节数据；第三字节“0x13”为读取数据命令；第四字节“0x00”为所读取的存储区为“00:Rsv”区；第五到第六字节“0x00 0x00”为读取到的数据。

5.3.2 操作步骤

（1）按照 5.1 节，配置、打开串口并进行标签识别。

（2）可以按照 5.2 节进行设置功率大小，功率设置大小范围为 5～20dBm。

（3）读取标签数据。

在读取数据前，选择其中一张识别到的标签，如图 5-26 所示。

图 5-26　寻卡操作

在标签读写区中，如图 5-27 所示。

图 5-27　存储区操作

① 在“安全”复选框中选择工作模式。

② 在存储区选择区中，共有 4 个存储区选择。

00：Rsv，保留内存存储区；保留一些特殊口令，无法修改，只能读取。

01：UII，EPC 存储区；包含标签的 UII 识别号等信息，一般不写入。

10：TID，是每张卡出厂时就设置好的唯一数据。无法修改，只能读取。

11：USER，用户区，在此存储区间用户可读可写。USER 区域数据存储长度为 32×2 字节。读取数据时，读取到的数据长度为“数量×2”字节长度（写入操作也一样），每个起始序号后带 2 个字节的数据长度。

③ 起始序号：默认为 3（可更改）。

④ 数量选择：默认为 1（可更改）。

⑤ 数据区域：显示读到或写入的数据，以十六进制数的形式显示。

在“非安全”模式下，在存储区分别选择 00:Rsv、01:UII、10:TID、11:USER，起始序号、数量均保持默认，分别进行读卡操作，读卡操作成功返回如图 5-28 所示的信息，返回的数据长度为 2 个字节。

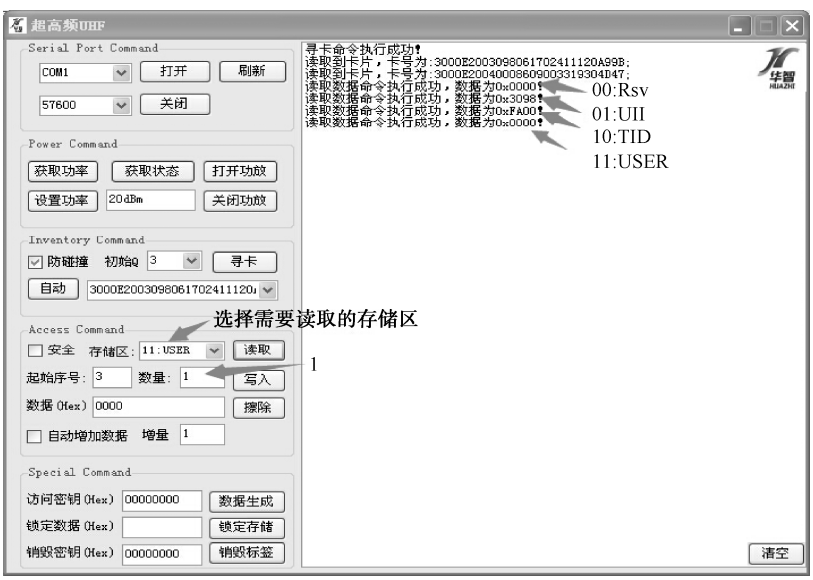


图 5-28　读取数据

在“非安全”模式下，在存储区分别选择 00:Rsv、01:UII、10:TID、11:USER，起始序号保持默认，数量将“1”改为“2”，分别进行读卡操作，读卡操作成功返回如图 5-29 所示的信

息，返回的数据长度为 4 个字节。

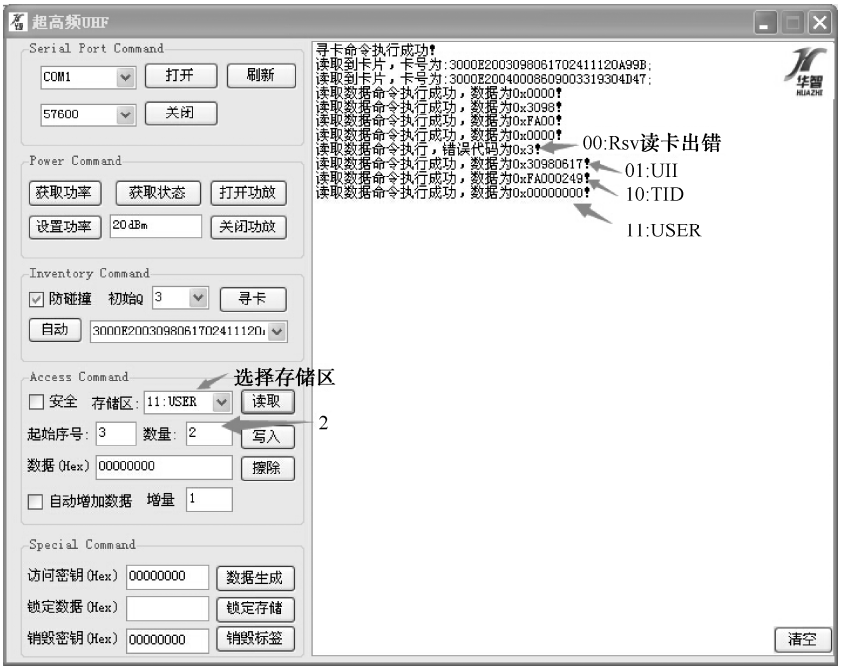


图 5-29　数据读取

在“安全”模式下，选中“安全”复选框，默认访问密钥为 0x00000000（可将访问密钥更改），如图 5-30 所示。若将密码改为 00000 或其他，再对 11:USER 用户存储区进行访问时就会读取数据出错。

图 5-30　数据读取失败

5.3.3 程序设计：读取数据功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境上创建 C#窗体应用程序；编写代码，实现数据读取功能。了解 Microsoft Visual Studio 2010 开发环境及窗体应用程序建立；掌握程序通过串口与 RFID 实验系统平台建立连接并通信的原理和方法；了解 RFID 实验平台 COM 协议并实现读取数据功能。

本设计所需设备如下。

（1）硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），ISO 18000-6（超高频 900MHz）RFID 原理模块（基于 32 位 ARM STM32 嵌入式处理器），ISO 18000-6 卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计主要原理：上位机应用程序采用 C#语言开发，遵守 C#编程规范，读取数据功能根据 RFID 超高频模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 RFID 超高频模块连接；上位机程序根据 RFID 超高频模块 COM 协议通过串口与 RFID 超高频模块进行通信，最终完成读取数据的功能。

2．设计步骤

在 5.2.3 节的基础上进行读取数据功能的实现。

第一步：在原有界面上进行界面设计。

在原有界面上拖入一个 GroupBox，将 Text 属性设置为“读写操作”；在“读写操作”的 GroupBox 中拖入一个 CheckBox，将 Text 属性设置为“安全”及 Name 属性设为“chk_UHF_Secured”；拖入四个 Label，将 Text 属性分别设置为“存储区：”、“起始序号：”、“数量：”和“数据（Hex）”；再拖入一个 ComboBox，将 Name 属性设置为“cmb_UHF_Bank”及 Items 属性设置为“00:Rsv”、“01:UII”、“10:TID”、“11:USER”，如图 5-31 所示。

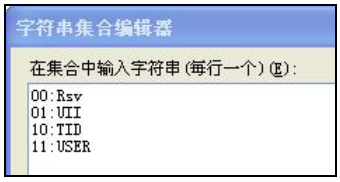


图 5-31　字符串集合编辑器

然后拖入 3 个 TextBox，适当调整大小，分别放到位置 1、位置 2 及位置 3 处，将 Name 属性分别设置为“txt_UHF_Ptr”、“txt_UHF_Count”及“txt_UHF_DATA”；将放置在位置 1 的 TextBox 的 Text 属性设置为“3”，将位置 2 的 TextBox 的 Text 属性设置为“1”，如图 5-32 所示。

在位置 3 下方拖入一个 Label 和一个 TextBox，将 Label 的 Text 属性设置为“访问密钥”，将 TextBox 的 Name 和 Txt 属性分别设置为“txt_UHF_AccessPwd”和“00000000”；最后拖出一个 Button，将 Name 属性设置为“btn_UHF_Read”及 Text 属性设为“读取数据”。

图 5-32　初步界面

最后设计好的界面如图 5-33 所示。

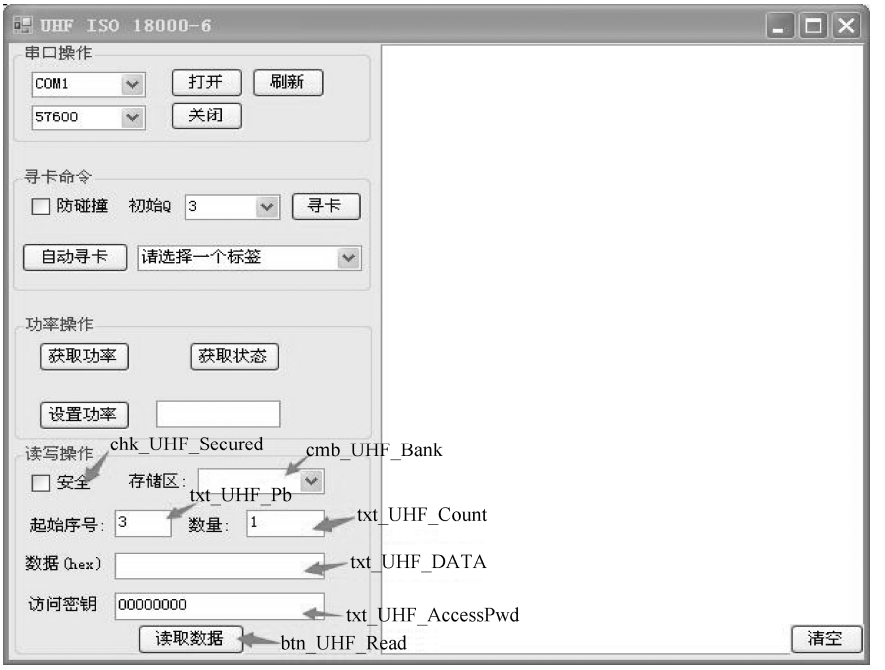


图 5-33　读取数据界面设计

第二步：编写代码实现功能（此处只附主要代码）。

UHF 数据读取功能执行过程如图 5-34 所示。

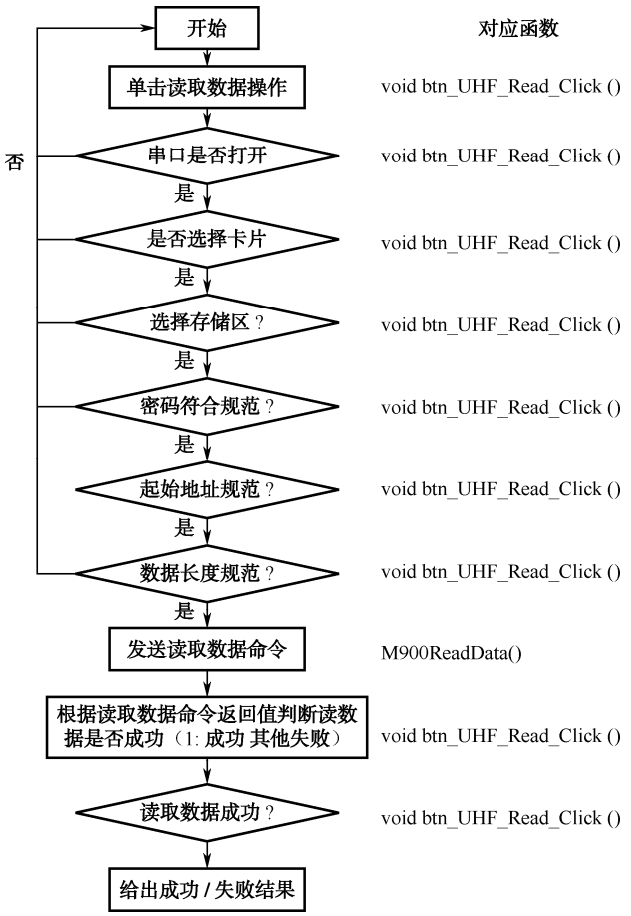


图 5-34　UHF 读取数据功能执行过程

部分代码解析如下：

```
//读卡事件
String struii = cmb_UHF_UII.Text.Trim();     //字符串保存卡号
Byte[] uUii = new Byte[struii.Length / 2];     //字节数组，长度为卡号长度的二分之一
for (Int32 i = 0; i < uUii.Length; i++)
        uUii[i] = Convert.ToByte(struii.Substring(i * 2, 2), 16);        //数据类型转换并保存
if (cmb_UHF_Bank.SelectedIndex < 0)                                      //判断是否选择读取的存储区
{
        AddUHFList("请选择需要读取的存储体！");
        return;                                                          //为选择则返回
}
Byte uBank = (Byte)cmb_UHF_Bank.SelectedIndex;                           //保存选择的存储体
String strpwd = txt_UHF_AccessPwd.Text.Trim();                           //字符串保存访问密钥
Byte[] uAccessPwd = new Byte[4];                                         //4 个字节长度的字节数组
if (chk_UHF_Secured.Checked)                                             //选中“安全”复选框
{
    try
    {
        for (Int32 i = 0; i < 4; i++)
```

```
            uAccessPwd[i] = Convert.ToByte(strpwd.Substring(i * 2, 2), 16);
                            //数据类型转换并保存到字节数组中
        }
        catch               //转换过程出现异常，给出异常原因
        {
            AddUHFList("您填写的访问密码不符合规范，将使用默认密码 0x00000000！");
            txt_UHF_AccessPwd.Text = "00000000";
            for (Int32 i = 0; i < 4; i++)
            uAccessPwd[i] = 0x00;
        }
    }
    else                    //未选中“安全”复选框，采用默认密码
    {
        uAccessPwd[0] = 0x00;
        uAccessPwd[1] = 0x00;
        uAccessPwd[2] = 0x00;
        uAccessPwd[3] = 0x00;
    }
    Byte[] uPtr = new Byte[1];                              //一个字节长度的字节数组
    try
    { uPtr[0]=Convert.ToByte(txt_UHF_Ptr.Text.Trim());}     //将起始序号数据类型转换并保存
    catch
    {
        AddUHFList("您填写的数据指针不符合规范，将使用默认值 0！");
        txt_UHF_Ptr.Text = "0";
        uPtr[0] = 0x00;
    }
    Byte uCnt = 0x01;
    try
    {
        uCnt=Convert.ToByte(txt_UHF_Count.Text.Trim());     //将“数量”数据类型转换并保存
        if (uCnt == 0x00)                                   //填写的数量不规范
        {
            AddUHFList("数据长度的值不符合规范，将使用默认值 1！");
            txt_UHF_Count.Text = "1";
            uCnt = 0x01;
        }
    }
    catch
    {
        AddUHFList("数据长度的值不符合规范，将使用默认值 1！");
        txt_UHF_Count.Text = "1";
    }
    Byte[] uReadData = new Byte[uCnt * 2];      //读取数据的长度为“数量”乘 2 个字节
    Byte[] uErrorCode = new Byte[1];            //读取过程出错的错误代码
    if (M900ReadData(hCom, uAccessPwd, uBank, uPtr, uCnt, uUii, uReadData, uErrorCode, 0) == 1)
    {   //函数功能：读取标签数据
        //返回值 1：成功读取标签数据；其他：读取标签数据失败
```

```
        //输入参数
        //HANDLE hCom：通信端口句柄
        //UCHAR* uAccessPwd：标签的 ACCESS PASSWORD
        //UCHAR uBank：标签的数据段类型
        //UCHAR* uPtr：起始地址的偏移量
        //UCHAR uCnt：读取数据的长度（两个字节为单位）
        //UCHAR* uUii：标签的 UII
        //UCHAR* uReadData：读取的标签数据，至少为 uCnt * 2 个字节
        //UCHAR* uErrorCode：只在函数返回失败，且 uErrorCode 不等于 0xFF 时有效
        //UCHAR  flagCrc：是否使用 CRC16 验证功能。1：使用 CRC 功能；0：不使用 CRC 功能
        String strdata = "";
        for (Int32 i = 0; i < uReadData.Length; i++)
        {   strdata += String.Format("{0:X2}", uReadData[i]); }
        AddUHFList(String.Format("读取数据命令执行成功，数据为 0x{0}！", strdata));
        txt_UHF_DATA.Text = strdata;              //显示读取数据
    }
    else                                          //读取失败
    {
        AddUHFList(String.Format("读取数据命令执行，错误代码为 0x{0}！", uErrorCode[0]));
    }
```

第三步：编译生成程序运行。

打开串口，然后识别标签，再选定标签，最后单击“读取数据”按钮，程序执行成功返回如图 5-35 所示的信息。

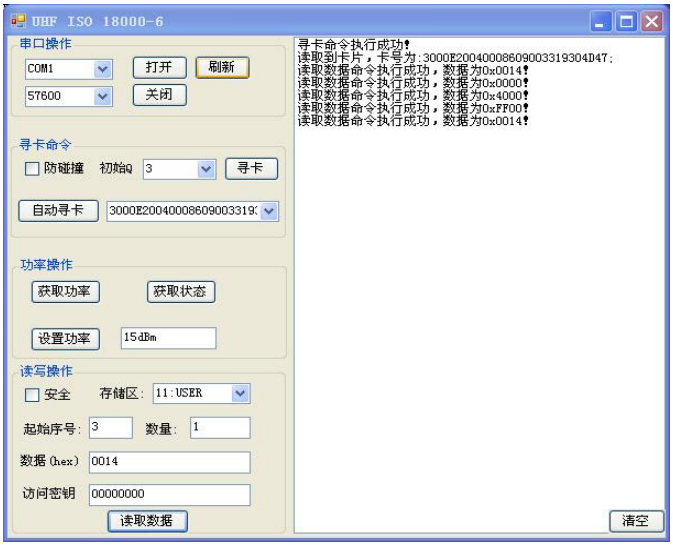


图 5-35　读取数据功能实现

5.4　UHF 900MHz ISO 18000-6 写入数据

5.4.1　协议原理

（1）ISO/IEC 18000-6 国际标准协议第六部分协议要求。

（2）写入数据。

功能介绍：当写入数量为 *i*（*i* 为大于 0 的正整数）时，该命令向指定 UII 的目标标签中写入 *i*×2 个字节长度的数据。

该功能操作有两种工作模式，分别是“非安全”、“安全”工作模式。

① 当“安全”复选框未被选中时，该操作工作在非安全工作模式下，此时用户读取电子标签的指定信息，将正常地进行读取标签存储区，不需要其他的验证。

② 当“安全”复选框被选中时，该操作工作在安全工作模式下，此时用户读取电子标签的信息，将需要密码验证，默认密码为 0x00000000。

（3）标签数据写入命令集。

命令帧格式（指定 UII）如下：

数据段	SOF	LEN	CMD	APWD	BANK	PTR	CNT	Data	UII	*CRC16	EOF
长度	1	EBV	1	4	1	EBV	1	CNT×2		2	1

数据段说明：

① APWD：标签的数据访问密码。

当用户读取的数据存储区为非 Reserved 存储区时，APWD 将置为 0x00000000。

当用户读取的数据存储区为 Reserved 存储区时，将 APWD 为标签的访问密钥。

② BANK：标签的存储分区。

Reserved 存储区，BANK=0x00。

UII 存储区，BANK=0x01。

TID 存储区，BANK=0x02。

USER 存储区，BANK=0x03。

③ PTR：标签存储区的起始地址，EBV 格式。

④ CNT：数据长度，以 Word（2B）为单位，支持 CNT=0。

⑤ Data：欲写入的数据，长度为 CNT×2 个字节。

⑥ UII：目标标签的 UII 数据。

（4）具体发送的数据帧命令：

写数据（卡号为“3000E2004000860900331930 4D47”，存储区为“11:USER”区，未选中“安全”复选框，起始序号为 3，数量为 1，写入数据为“0004”）

发送命令：

```
AA 19 14 00 00 00 00 03 03 01 00 04 30 00 E2 00 40 00 86 09 00 33 19 30 4D 47 55
```

数据命令解析：第一字节“0xAA”为数据帧开始标识，最后一个字节“0x55”为数据帧结束标识；第二字节“0x19”（转换成十进制数为 25）为从数据帧开始标识 0xAA 到数据帧结束标识 0x55 之间共有 25 个字节数据；第三字节“0x14”为写入数据命令；第四到第七字节“0x00 0x00 0x00 0x00”为标签的数据访问密码；第八字节“0x03”为要读取的存储区为 User 区（Rsv 存储区：0x00；UII 存储区：0x01；TID 存储区：0x02；USER 存储区：0x03）；第九字节“0x03”为标签存储区的起始地址；第十字节“0x01”为写入数据的数量为 1×2 字节；第十一、十二字节“0x00 0x04”为要写入的数据。第十三字节到第二十六字节为要进行数据读取的标签号。

命令返回：

AA 03 14 00 55(返回的结果为：写入成功)

5.4.2 操作步骤

（1）按照 5.3 节的步骤读出数据。

（2）在如图 5-36 所示的区域中进行参数的设置。

图 5-36　写入数据

在存储区选择要写入数据信息的存储区。

设置写入数量为“1”，即 2×1 个字节长度。

在数据区输入要写入的数据信息，输入的数据为十六进制数。

（3）对每个存储区都分别进行写入数据操作，返回执行结果如图 5-37 所示。

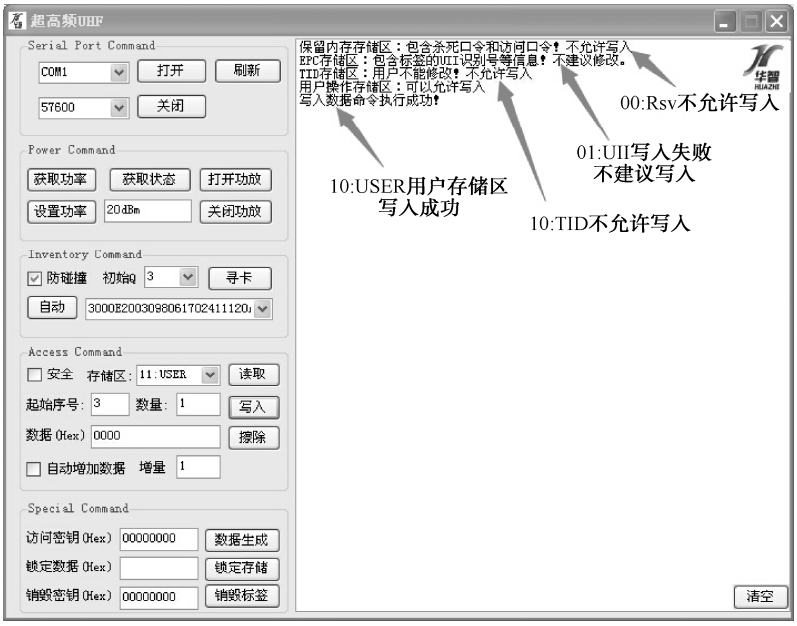


图 5-37　写入数据成功

（4）写入数据自动增加，存储区选择“11:USER”用户区，选中“自动增加数据”复选框，其他参数保持默认。初始数据为 0x0000，每写入一次，写入数据将按增量（现为 1）增加，写入的数据为 0x0000、0x0001、…、0x0005，如图 5-38 所示。

在“安全”模式下进行写入数据操作，同样写入数据也需要访问密钥，且密钥与默认密钥不符时将会提示出错，指令执行结果如图 5-39 所示。

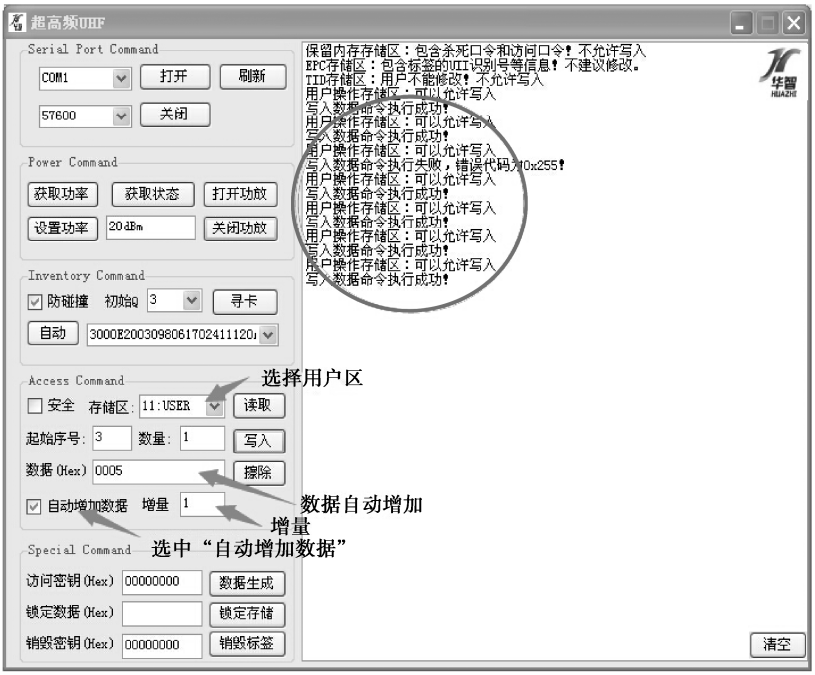


图 5-38　自动增加写入操作

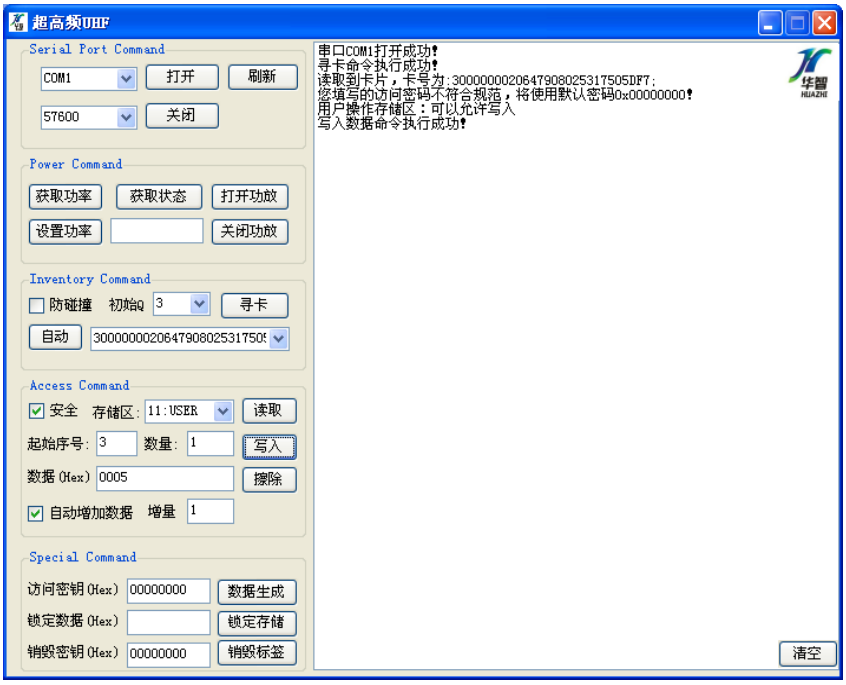


图 5-39　安全模式下写入

从图 5-39 中可以看出，在“安全”模式下，访问密钥与默认密钥不符时，执行返回结果会提示访问密钥不符合规范，并修改访问密钥栏的访问密钥。

5.4.3 程序设计：写入数据功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境下创建 C#窗体应用程序；编写代码，实现写入数据功能。主要目的是了解 Microsoft Visual Studio 2010 开发环境及窗体应用程序建立；掌握程序通过串口与 RFID 实验系统平台建立连接并通信的原理和方法；了解 RFID 实验平台 COM 协议并实现写入数据功能。

本设计所需设备如下。

（1）硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），ISO 18000-6（超高频 900MHz）RFID 原理模块（基于 32 位 ARM STM32 嵌入式处理器），ISO 18000-6 卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计主要原理：上位机应用程序采用 C#语言开发，遵守 C#编程规范，写入数据功能根据 RFID 超高频模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 RFID 超高频模块连接；上位机程序根据 RFID 超高频模块 COM 协议通过串口与 RFID 超高频模块进行通信，最终完成写入数据的功能。

2．设计步骤

在 5.3.3 节的基础上实现写入数据功能。

第一步：界面设计。

拖入一个新的 Button 按钮放置在“读取数据”按钮的右边，将 Button 按钮 Name 属性设置为“btn_UHF_Write”，将 Text 属性设置为“写入数据”，然后双击该按钮添加按钮响应事件。

完成上述步骤后的界面如图 5-40 所示。

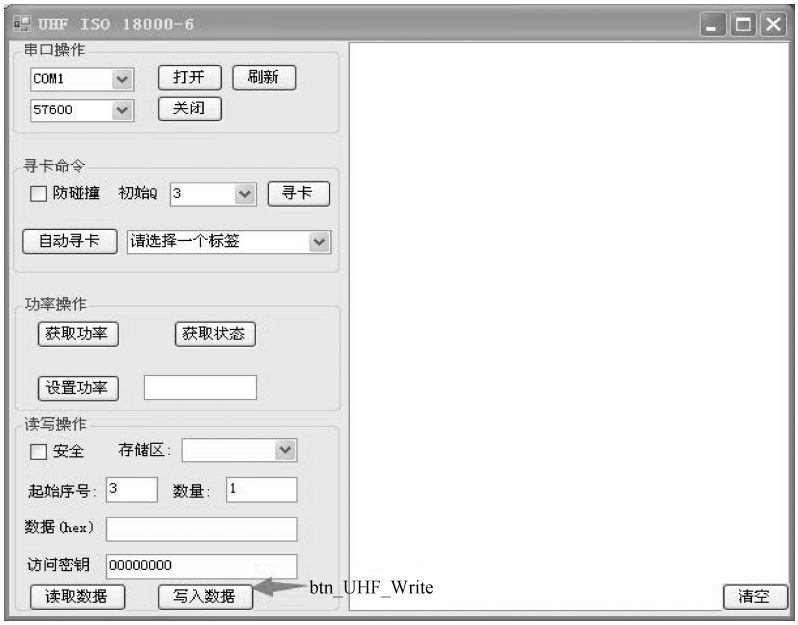


图 5-40　写入数据界面设计

第二步：编写代码实现功能（此处只附部分主要代码）。

UHF 写入数据功能执行过程如图 5-41 所示。

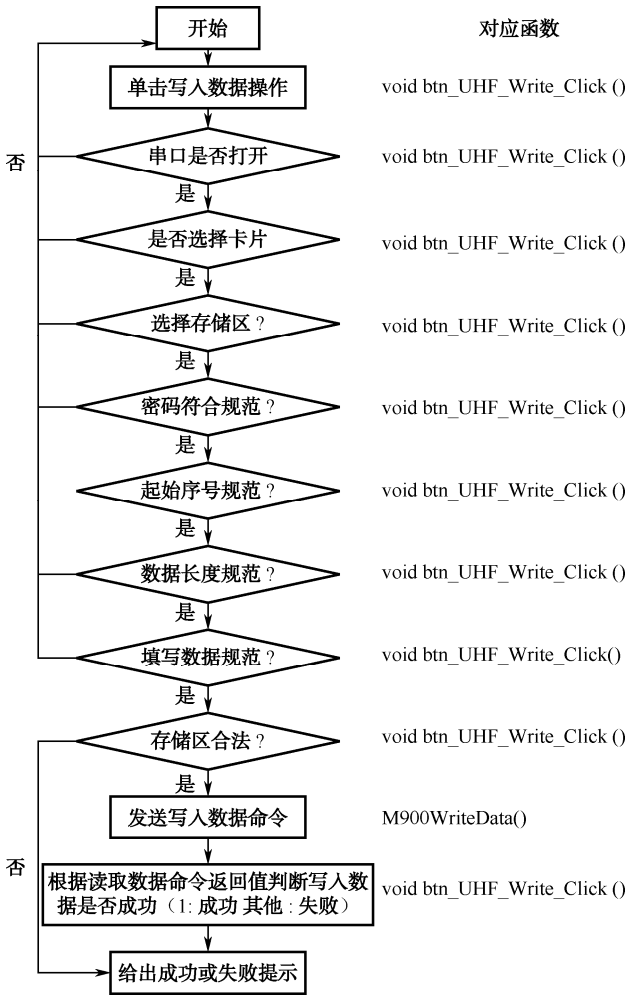


图 5-41　UHF 写入数据功能执行过程

部分代码解析如下：

```
//写入数据事件
String struii = cmb_UHF_UII.Text.Trim();                    //用字符串保存卡号
Byte[] uUii = new Byte[struii.Length / 2];                  //字节数组保存卡号
for (Int32 i = 0; i < uUii.Length; i++)
{
    uUii[i] = Convert.ToByte(struii.Substring(i * 2, 2), 16);    //类型转换并保存
}
if (cmb_UHF_Bank.SelectedIndex < 0)                         //判断是否选择存储区
{
    AddUHFList("请选择需要写入的存储体！");
    return;
}
Byte uBank = (Byte)cmb_UHF_Bank.SelectedIndex;              //保存写入的存储区
```

```
String strpwd = txt_UHF_AccessPwd.Text.Trim();                    //字符串保存访问密码
Byte[] uAccessPwd = new Byte[4];                                  //字节数组保存
if (chk_UHF_Secured.Checked)                                      //判断是否选中安全状态
{
    try
    {
        for (Int32 i = 0; i < 4; i++)
        uAccessPwd[i] = Convert.ToByte(strpwd.Substring(i * 2, 2), 16);
                                        //访问密码数据类型转换并保存
    }
    catch //填写的密码有误
    {
        AddUHFList("您填写的访问密码不符合规范，将使用默认密码 0x00000000！");
        txt_UHF_AccessPwd.Text = "00000000";
        for (Int32 i = 0; i < 4; i++)
        uAccessPwd[i] = 0x00;
    }
}
else //默认密码
{
    uAccessPwd[0] = 0x00;
    uAccessPwd[1] = 0x00;
    uAccessPwd[2] = 0x00;
    uAccessPwd[3] = 0x00;
}
Byte[] uPtr = new Byte[1];                                        //起始序号
try
{   uPtr[0] = Convert.ToByte(txt_UHF_Ptr.Text.Trim()); }
catch
{
    AddUHFList("您填写的数据指针不符合规范，将使用默认值 0！");
    txt_UHF_Ptr.Text = "0";
    uPtr[0] = 0x00;
}
Byte uCnt = 0x01;
try
{
    uCnt = Convert.ToByte(txt_UHF_Count.Text.Trim());       //保存写入的数据长度
    if (uCnt == 0x00)
    {
        AddUHFList("数据长度的值不符合规范，将使用默认值 1！");
        txt_UHF_Count.Text = "1";
        uCnt = 0x01;
    }
}
catch
{
    AddUHFList("数据长度的值不符合规范，将使用默认值 1！");
```

```
        txt_UHF_Count.Text = "1";
    }
    String strwritedata = txt_UHF_DATA.Text.Trim();
    if (strwritedata.Length != (uCnt * 4))
    {
        AddUHFList("实际数据长度与您填写的数据长度的值不符！");
        AddUHFList(String.Format("请填写 2*{0}个字节的十六进制数据！", uCnt));
        txt_UHF_DATA.SelectAll();
        txt_UHF_DATA.Focus();
        return;
    }
    Byte[] uWriteData = new Byte[uCnt * 2];
    try
    {
        for (Int32 i = 0; i < uWriteData.Length; i++)
        {
            uWriteData[i] = Convert.ToByte(strwritedata.Substring(i * 2, 2), 16);
        }
    }
    catch
    {
        AddUHFList("您填写的数据有误！");
        AddUHFList(String.Format("请填写 2*{0}个字节的十六进制数据！", uCnt));
        txt_UHF_DATA.SelectAll();
        txt_UHF_DATA.Focus();
        return;
    }
    Byte[] uErrorCode = new Byte[1];
    if (uBank == 0x03)
    {
        AddUHFList(String.Format("用户操作存储区：可以允许写入"));
        if (M900WriteData(hCom, uAccessPwd, uBank, uPtr, uCnt, uUii, uWriteData, uErrorCode, 0) == 1)
        //函数功能：写入数据标签 写入成功
        {
            //返回值 1：成功写入标签数据。其他：写入标签数据失败
            //输入参数 HANDLE hCom：通信端口句柄
            //UCHAR* uAccessPwd：标签的 ACCESS PASSWORD
            //UCHAR uBank：标签的数据段类型
            //UCHAR* uPtr：起始地址的偏移量
            //UCHAR uCnt：写入数据的长度（两个字节为单位）
            //UCHAR* uUii：标签的 UII
            //UCHAR* uWriteData：需要写入的数据
            //UCHAR* uErrorCode：只在函数返回失败，且 uErrorCode 不等于 0xFF 时有效
            //UCHAR  flagCrc：1：使用 CRC 功能；0：不使用 CRC 功能
            AddUHFList(String.Format("写入数据命令执行成功！")); //写入成功
            strwritedata = strwritedata.Remove(strwritedata.Length - 2, 2);
            strwritedata += String.Format("{0:X2}", uWriteData[uWriteData.Length - 1]);
            txt_UHF_DATA.Text = strwritedata.ToUpper();                    //显示写入数据
```

```
            }
            else
            {
                AddUHFList(String.Format("写入数据命令执行失败，错误代码为 0x{0}！", uErrorCode[0]));
            }
        }
        else //写入其他区，其他存储区不允许写入
        {
            if (uBank == 0x00)
            {
                AddUHFList(String.Format("保留内存存储区：包含删除口令和访问口令！不允许写入"));
            }
            if (uBank == 0x01)
            {
                AddUHFList(String.Format("EPC 存储区：包含标签的 UII 识别号等信息！不建议修改。"));
            }
            if (uBank == 0x02)
            {
                AddUHFList(String.Format("TID 存储区：用户不能修改！不允许写入"));
            }
        }
```

第三步：编译生成程序运行，运行结果如图 5-42 所示。

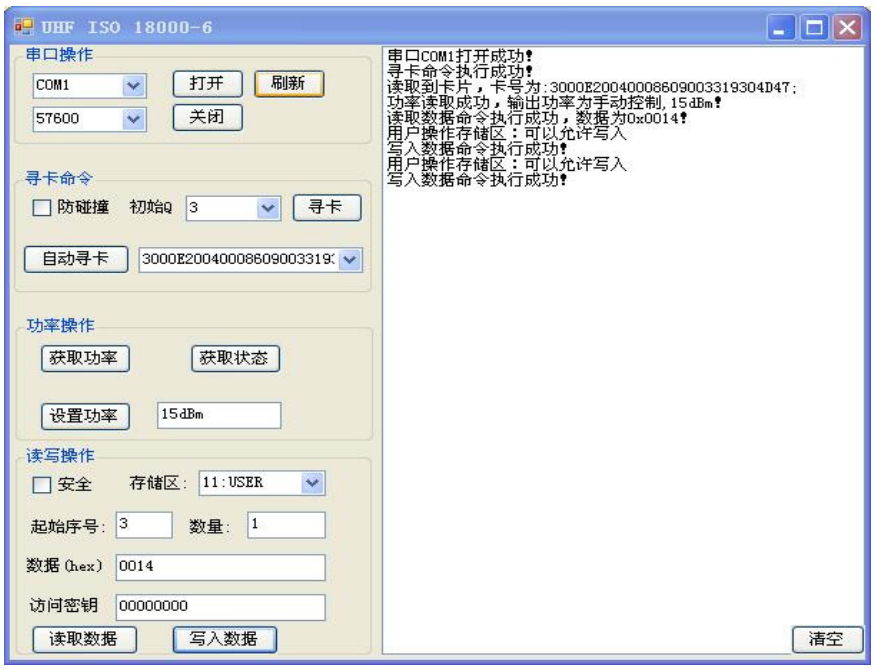


图 5-42　写入数据实现

第 6 章　2.4G 有源 RFID 协议原理及实践开发

根据实现方式的不同，RFID 可分为两类：有源 RFID 和无源 RFID。无源 RFID 的电子标签上不带电池，其工作所需要的全部电源都依靠转接收到的阅读器发送的电磁波而获得，所以其阅读器的发射功率一般较大。与之相反，有源 RFID 的电子标签自身具备电池，可提供全部器件工作的电源，因而相应阅读器的发射功率要求不高，而且有效阅读距离也较前者有所增加。

本章所述内容所使用设备包括桂林华智 RFID 物联网教学科研平台实验箱 2.4G 有源读写器、2.4G 有源标签卡片、串口线。主要操作内容：通过使用上位机软件，与 2.4G 有源 RFID 模块进行通信，完成对 2.4G 有源标签的操作。通过本章的学习，掌握对 2.4G 有源读写器的操作，了解相关协议及工作原理。

（1）2.4G 有源读写器工作电压为 5V，工作温度为 0℃～60℃，工作频段为 HF 2.4GHz。这种读写器适用于多种场合读写卡应用，广泛地应用到公路收费、港口货运管理等应用中，具有广阔的应用前景。

（2）阅读器平面图，如图 6-1 所示。

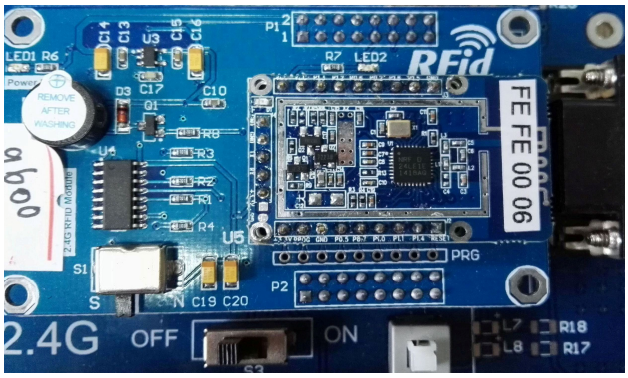


图 6-1　阅读器平面图

（3）阅读器电路图如图 6-2 所示。

（4）工作原理简介。

有源 RFID 系统主要包括上位机、阅读器、有源标签三大部分，如图 6-3 所示。其中上位机就是普通计算机，是 RFID 识别系统与特定应用系统的连接点。阅读器在系统中的作用是探测监听附近区域的标签，解析并存储其 ID，等待主机查询取用。有源 RFID 系统中的阅读器与无源系统的阅读器在原理和结构上没有本质区别。有源标签的最主要特点是标签不依靠阅读器发送的载波提供能量，而是具有独立的能量供应系统。所以有源标签与无源标签相比，具有识别距离更远，配套阅读器发射功率更低的优点，但也有标签成本高、体积大、寿命短等缺点。

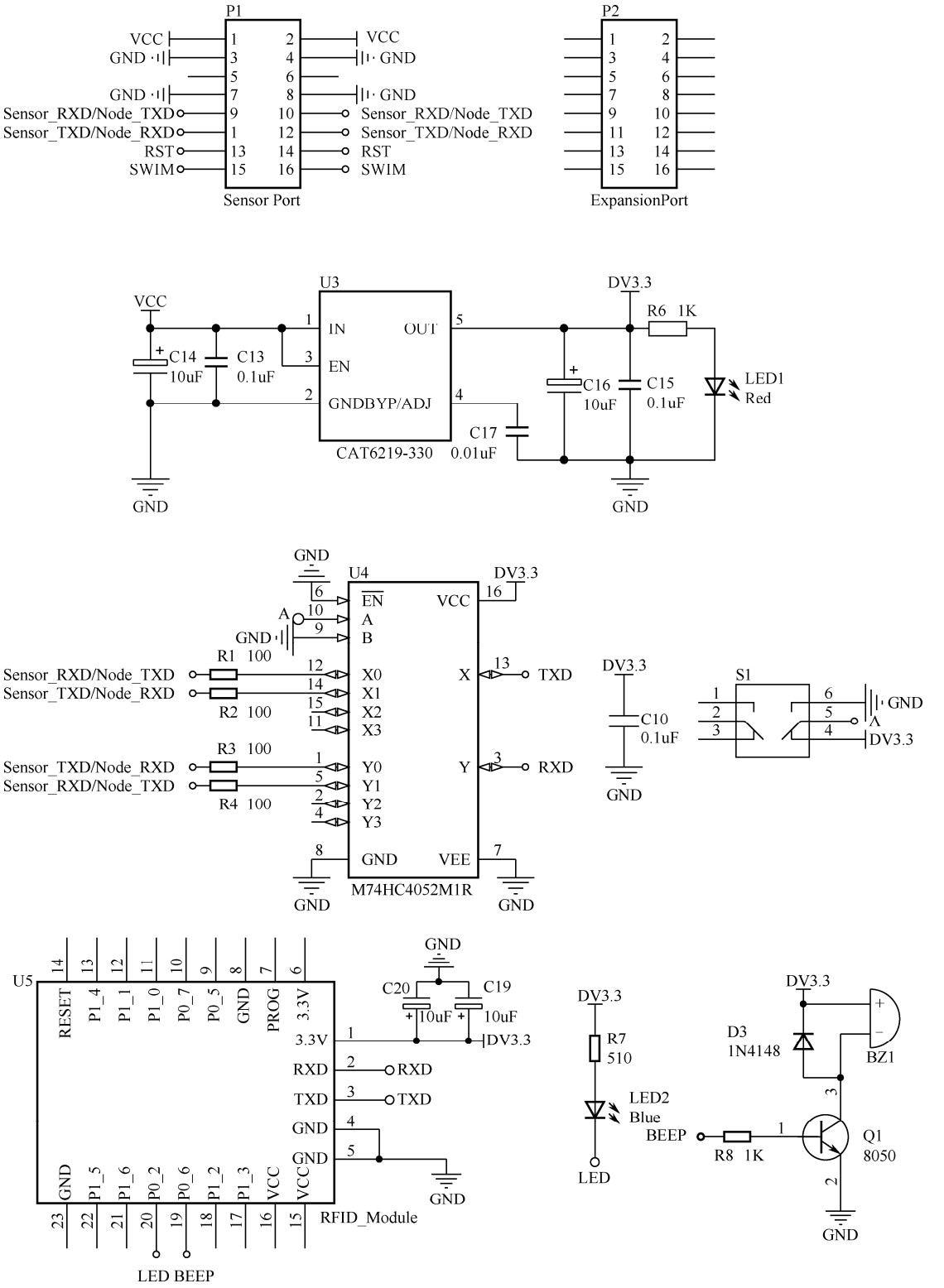

P1
VCC
GND
GND
Sensor_RXD/Node_TXD
Sensor_TXD/Node_RXD
RST
SWIM
Sensor Port
P2
ExpansionPort
U3
CAT6219-330
C14 10uF
C13 0.1uF
C17 0.01uF
C16 10uF
C15 0.1uF
R6 1K
LED1 Red
DV3.3
U4
M74HC4052M1R
R1 100
R2 100
R3 100
R4 100
C10 0.1uF
S1
TXD
RXD
U5
RFID_Module
C20 10uF
C19 10uF
R7 510
LED2 Blue
R8 1K
Q1 8050
D3 1N4148
BZ1
LED BEEP

图 6-2　阅读器电路图

（5）标签工作机制原理。

在该工作机制中定义了标签的 4 种工作状态：休眠状态、信道查询状态、半休眠状态、通信状态，如图 6-4 所示。

① 休眠状态：是指除定时器外，标签的所有部件均停止工作。

② 信道查询状态：是指标签被某事件唤醒后，查询信道上的有效阅读器信号。

③ 半休眠状态：如果与其他标签发生碰撞，暂时休眠一段事件。

④ 通信状态：建立与阅读器的有效连接，实现数据的传输。

大多数情况下，标签处在休眠状态。此时，标签上几乎所有的部件均停止工作，但定时器正常工作。当其计数到休眠唤醒时间后，将标签唤醒，进入信道查询状态。

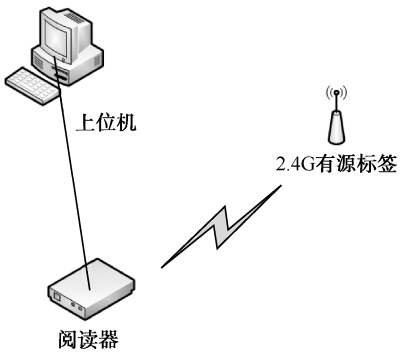


图 6-3　有源 RFID 系统的组成

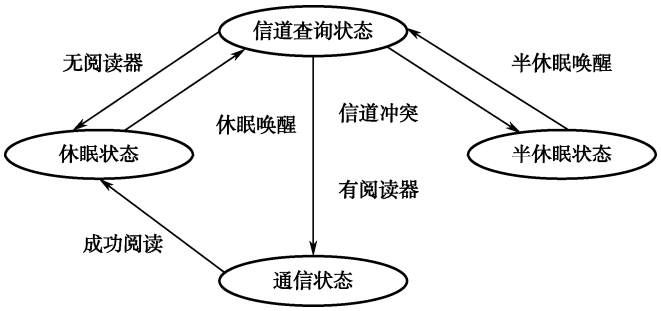


图 6-4　标签工作机制原理

（6）低功耗的设计。在有源 RFID 系统中，如何降低标签的功耗是一个关键技术。总结前面内容中相关的部分，大致有以下几点。

① 信号的调制方式是 O-QPSK 且 I、Q 通道同时使用，在相同信息的条件下，较二进制传输的传输时间缩短了，减小了功耗。

② 采用扩频技术缩短了同步时间。

③ 半双工的工作方式，减少了功耗。

④ 由于调制方式相对简单，因此相应的电路功耗较小。

（7）读写器主芯片介绍。2.4G 主芯片 NRF24LE1 采用了 NORDIC 最新的无线和超低功耗技术，在一个极小封装中集成了包括 2.4G 无线传输，增强型 51 Flash 高速单片机，丰富外设及接口等的单片 Flash 芯片，适合应用于各种 2.4G 的产品设计。

NRF24LE1=2.4GHz+Flash51+ADC+PWM+I2C+RTC+WDT+RNG+AES+COMP+UART+SPI

NRF24LE1E 基本特性如下。

① nRF24LE1 模块是采样 NORDIC-2.4GHz 频段（微波）无线收发，采用 GFSK（高斯频移键控，Gauss Frequency Shift Keying，在调制之前通过一个高斯低通滤波器来限制信号的频谱宽度）调制，2Mbps 空中速率无线收发（双向）传输数据。

② 增强型 C51 为内核，数倍于标准 51 的速度，时钟频率最高 16MHz，系统工作电压低、低功耗等特点。

③ nRF24LE1 多 IO 口（20，32，48PIN，QFN 封装）。

④ 传输距离 30m 内。

GFSK 高斯频移键控调制是把输入数据经高斯低通滤波器预调制滤波后，再进行 FSK 调制的数字调制方式。它在保持恒定幅度的同时，能够通过改变高斯低通滤波器的 3dB 带宽对已调信号的频谱进行控制，具有恒幅包络、功率谱集中、频谱较窄等无线通信系统所希望的特性。GFSK 调制可以分为直接调制和正交调制两种方式。

NRF24LE1E 主要特性如下。

① 内嵌 2.4GHz 低功耗无线收发内核 nRF24L01P，250kbps、1Mbps、2Mbps 空中速率。

② 高性能 51 内核（12 倍工业标准 51 的速度），16KB Flash，1 KB data RAM，1KB NV data RAM。

③ 具有丰富的外设资源，内置 128 位 AES 硬件加密，32 位硬件乘除处理器，6～12 位 ADC，两路 PWM，I^2C，UART，硬件随机数产生器件，WDT，RTC，模拟比较器。

④ 灵活高效的开发手段，支持 Keil C、ISP 下载，是开发无线外设、RFID、消费产品等有力的工具及平台。

（8）应用领域：无线鼠标，无线键盘，无线摇杆，玩具，RFID，无线遥控。

6.1 2.4G 有源 RFID 寻卡操作及实现

6.1.1 协议原理

（1）典型的有源 RFID 识别系统结构的组成主要包括上位机、阅读器、有源标签三大部分。上位机通过向阅读器发送指令，阅读器接收并执行来自上位机的指令，并监听附近区域的标签，对标签进行操作，将执行指令后的结果发送给上位机。

（2）在通信时，数据帧结构如下：

Char	Len	Addr	Attr	Type	Info	XOR
6 B	2 B	4 B	2 B	1 B	*x* B	1 B

Char：帧头，6B，固定值 0xFFFFFFFFFF00，后续所有字段确保不会出现帧头序列。

Len：字节长度，2B：Addr+Attr+Type+Info+XOR。

Addr：读写模块 ID，4B，0xFFFFFFFE 为广播 ID，下发指令时可用广播 ID。

Attr：读写模块属性，2B，保留。

Type：类型字，1B，见后面详细定义。

Info：与类型对应，见后面详细定义。

XOR：字节异或校验，1B，用于数据帧校验，从 Len 到 Info 逐字节异或。

“搜索指令”下行：Type=0x18，Info 0B。

字　节	位	取值范围	备　注
Type	7～0	0x18	搜索指令

搜索指令在下发后 2 秒内获得所有在场的标签信息。

“搜索指令的回复/自动上报”上行：Type=0x00，Info 长度与标签数有关（标签数×16）。

字　　节	位	取 值 范 围	备　　注	
Type	7～0	0x00	搜索指令的回复/自动上报	
Info[0～3]			标签 ID	标签 1 信息 长度 16B：4B 的 ID+2B 的状态+10B 的携带
Info[4]	7	0/1	低电指示：1—低电	
	6～4	0～2	传感类型：0—NFC，1—无，2—温度	
	3～0	0	传感数据（1/2B）	
Info[5]	7～0	0	传感数据（1B）	
Info[6～15]			10 字节携带（取自内部存储区）	
Info[16～X]			更多标签信息，定义同标签 1	每标签 16B

每标签 16B，读写模块的标签容量是 60 个。

2.4G 寻卡发送：

```
FF FF FF FF FF 00 00 08 FF FF FF FE 00 00 18 11 (16B)
```

其中，“FF FF FF FF FF 00”为帧头，“08”表示数据“FF FF FF FE 00 00 18 11”的长度，“11”为 CRC 校验。

寻卡数据返回：

```
FF FF FF FF FF 00 00 18
FE FE 00 06 00 00 00 02 00 00 21 10 00 00 00 00 00 00 00 00 00 00 00 2D (卡号为 02000021)
```

（3）读写模块工作特征

读写模块的标签容量为 60 个。可以自动上报，也可以接收搜索指令，2 秒后回复此间收到的标签信息。设置为自动上报时，若现场无标签，则不输出；若存在标签，则以 2 秒为周期定时输出。在关闭自动上报的情况下，可以用搜索指令按需要查询在场的标签。注意搜索指令是在指令下发 2 秒后获得回复。

读写模块除了接收和输出标签信息，还处理 UART 端口收到的面向模块本身或指定标签的存储访问指令：读、写、擦。指令合法并执行成功，会有返回信息，否则得不到返回信息。

标签的工作机制是：1 秒发射一次，每 10 次携带 1 个接收窗口。

6.1.2　操作步骤

（1）打开 RFID 电源，用串口线建立通信连接。

（2）将 S1 开关拨至 S。

（3）打开上位机软件，选择正确的串口号和默认的波特率 9600（如果选择其他波特率，可能导致无法正常识别标签），如图 6-5 所示。

（4）打开串口。在对串口进行配置完成后，便可以打开串口进行通信，如图 6-6 所示。

（5）寻卡操作。在打开串口以后，上位机与下位机便完成了通信连接，此时上位机就可以给下位机发送寻卡指令进行寻卡操作，寻卡成功返回如图 6-7 所示的信息。

图 6-5　界面介绍

图 6-6　打开串口

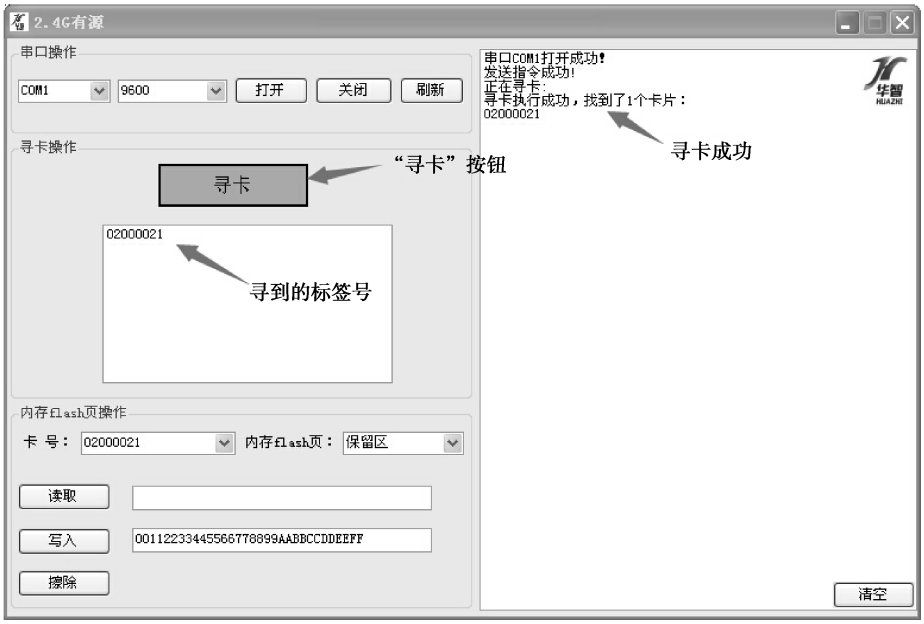


图 6-7　寻卡成功

6.1.3　程序设计：寻卡功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境下创建 C#窗体应用程序；编写代码，实现寻卡功能。主要目的是了解 Microsoft Visual Studio 2010 开发环境及窗体应用程序建立；掌握程序通过串口与 RFID 实验系统平台建立连接并通信的原理和方法；了解 RFID 实验平台 COM 协议并实现寻卡功能。

本设计所需设备如下。

（1）硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），2.4G RFID 原理模块（基于 32 位 ARM STM32 嵌入式处理器），2.4G 模块卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计主要原理：上位机应用程序采用 C#语言开发，遵守 C#编程规范，寻卡功能根据 RFID 2.4G 模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 RFID 2.4G 模块连接；上位机程序根据 RFID 2.4G 模块 COM 协议通过串口与 RFID 2.4G 模块进行通信，最终完成寻卡的功能。

2．设计步骤

第一步：创建 C#窗体应用程序。

启动 Microsoft Visual Studio 2010 开发平台，创建一个如图 6-8 所示的 C#窗体应用程序，命名为“2.4G_findCard”，创建的详细方法可参考第 3 章。

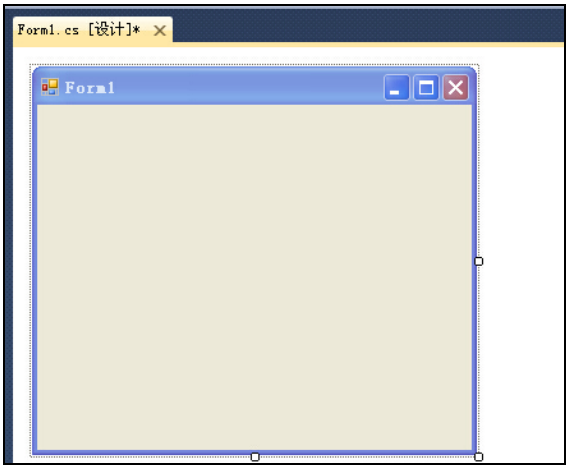


图 6-8　新建窗体

第二步：设计功能界面。

（1）选中新建的窗体，右击界面，在弹出的快捷菜单中选择“属性”选项，将 Text 属性改为“2.4G_findCard”。

（2）打开工具箱  ，拖出一个 GroupBox，将 Text 属性改为“串口操作”，拖出两个 ComboBox 放到串口操作的 GroupBox 中，并将其 Text 属性分别改为“串口号”与“波特率”，并将 ComboBox 与“串口号”对应的 Name 属性改为“cmb_PortNum_2p4G”，将 ComboBox 与“波特率”对应的 Name 属性改为“cmb_BaudRate_2p4G”，并将 Items 属性设置

为 9600、57600、115200；拖出三个 Button 放到 GroupBox 中，分别将其 Text 属性设置为“打开”、“关闭”、“刷新”，Name 属性分别对应为“btn_Open_2p4G”、“btn_Close_2p4G”、“btn_Refresh_2p4G”。

（3）再拖出一个 GroupBox，将 Text 属性改为“寻卡操作”，将 Name 属性设置为“groupBox_Inventory”，Enabled 属性设置为 False；拖出一个 Button 放到寻卡操作的 GroupBox 中，将其 Text 属性设置为“寻卡”，将 Name 属性设置为“btn_Inventory_2p4G”；拖出一个 listBox 控件放到寻卡操作的 GroupBox 中，将 Name 属性设置为“lst_InventoryResult_2p4G”。

（4）拖出一个 ListBox 控件放在界面右侧，将其 Name 属性设置为“lsb_Info_2p4G”，拖出一个 Button 放到其右下方，将 Name 属性设置为“btn_Clear_2p4G”，Text 属性为“清空”。

完成后的寻卡界面如图 6-9 所示。

图 6-9　寻卡界面

第三步：编写代码实现功能（此处只附部分主要代码）。

2.4G 寻卡功能执行过程如图 6-10 所示。

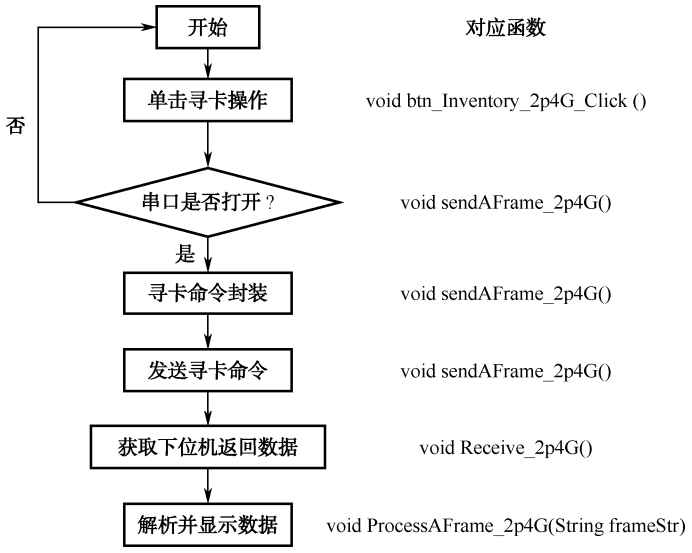


图 6-10　2.4G 寻卡功能执行过程

部分代码解析如下：

```
//寻卡事件
private void btn_Inventory_2p4G_Click(object sender, EventArgs e)
{
    CurrCMD_2p4G = SEARCHCARD_2P4G;
    // private const int SEARCHCARD_2P4G = 0;定义常量，标明当前操作为寻卡操作
    byte[] len ={0x00,0x08};    //命令中有效数据（除去帧头和表示长度的字节）长度为 8 字节
    byte[] addr = { 0xFF, 0xFF, 0xFF, 0xFE };          //地址
    byte type = 0x18;                                  //命令类型：寻卡
    sendAFrame_2p4G(len, addr, type, null);            //发送一帧寻卡命令
    ShowInfo2p4G("正在寻卡:");                          //将信息显示到信息栏
}
//2.4G 发送一帧数据的方法
private void sendAFrame_2p4G(byte[] len, byte[] addr, byte type, byte[] info)
{
    if (!IsOpen_2p4G)                                  //判断串口是否打开
    {
        ShowInfo2p4G(String.Format("错误：请先通过串口连接设备！"));
        groupBox_Inventory.Enabled = false;
        groupBox_flash.Enabled = false;
        return;                    //如果串口未打开，则跳出函数，不再往下执行
    }
    int dataLen = len[0] * 256 + len[1] + 8;           //获取数据帧长度
    byte[] frame = new byte[dataLen];
    frame[0] = 0xFF;frame[1] = 0xFF; frame[2] = 0xFF;frame[3] = 0xFF;//帧头
    frame[4] = 0xFF;frame[5] = 0x00;                   //帧头
    frame[6] = len[0];frame[7] = len[1];      //有效数据长度，保存在数据帧第 7、8 字节中
    frame[8] = addr[0];frame[9] = addr[1];frame[10] = addr[2];frame[11] = addr[3];
    //地址
    frame[12] = 0x00;
    frame[13] = 0x00;
    frame[14] = type;              //数据帧命令类型
    if (info != null)              //判断 info 字段是否为空，如果不为空，则添加到数据帧中
    {
        int infoLen = info.Length;
        for (int i = 0; i < infoLen; i++)
        {
            frame[15 + i] = info[i];
        }
    }
    int crc = 0;                   //整型变量，用于 CRC 校验
    for (int i = 6; i < dataLen - 1; i++)
    {
        crc = crc ^ frame[i];      //计算 CRC 校验
    }
    frame[dataLen - 1] = (byte)crc;                    //将 CRC 校验结果放在数据帧最后一个字节
    try
```

```
            {
                serialPort_2p4G.Write(frame, 0, dataLen);       //向串口发送一帧数据命令
                ShowInfo2p4G("发送指令成功!");
                //CurrCMD_2p4G = type;
            }
            catch (Exception ex)                                //发送命令异常，并捕获异常
            {
                ShowInfo2p4G(String.Format("错误：数据发送异常！错误信息为:{0}", ex.Message));
                IsOpen_2p4G = false;
            }
        }
        /*其他函数声明及功能（具体函数定义见源代码）
        获取串口的方法：public Int32 GetComList(out List<String> ComList) 功能：获取本机串口
        显示信息函数：private void ShowInfo2p4G(String msg) 功能：将提示信息显示在信息框处
        打开串口方法：private void btn_Open_2p4G_Click(object sender, EventArgs e) 功能：打开串口
        关闭串口方法：private void btn_Close_2p4G_Click(object sender, EventArgs e) 功能：关闭串口
        2.4G 接收数据的方法：private void Receive_2p4G() 功能：接收下位机发送的数据
        2.4G 解析一帧数据方法：private void processAFrame_2p4G(String frameStr)功能：对接收的数据进
        行解析，参数 String frameStr 为接收数据的方法接收到的数据
        */
```

第四步：编译生成程序运行。

（1）在菜单中选择“调试”→“启动调试”选项或按 F5 键生成程序并运行。

（2）选择正确的波特率和串口号，单击“打开”按钮，串口打开成功后单击“寻卡”按钮，执行寻卡操作，结果如图 6-11 所示。

图 6-11　寻卡实现

6.2　2.4G 有源 RFID 读取数据操作及实现

6.2.1　协议原理

（1）通过使用上位机软件与 RFID 高频 2.4G 模块下位机进行通信，完成对 2.4G 有源标签的读取数据。

（2）典型的有源 RFID 识别系统结构的组成主要包括上位机、阅读器、有源标签三大部分。上位机通过向阅读器发送指令，阅读器接收并执行来自上位机的指令，并监听附近区域的标签，对标签进行操作，将执行后指令的结果发送给上位机。

（3）在通信时，数据帧结构如下：

Char	Len	Addr	Attr	Type	Info	XOR
6 B	2 B	4 B	2 B	1 B	*x* B	1 B

① Char：帧头，6B，固定值 0xFFFFFFFFFF00，后续所有字段确保不会出现帧头序列。

② Len：字节长度，2B，Addr+Attr+Type+Info+XOR。

③ Addr：读写模块 ID，4B。

0xFFFFFFFE 为广播 ID，下发指令时可用广播 ID。

④ Attr：读写模块属性，2B，保留。

⑤ Type：类型字，1B，见后面详细定义。

⑥ Info：与类型对应，见后面详细定义。

⑦ XOR：字节异或校验，1B，用于数据帧校验，从 Len 到 Info 逐字节异或。

访问标签内存：Type=0x8A，Info 24B。

字　节	位	取值范围	备　注	
Type	7～0	0x8A	访问标签	
Info[0～3]			目标标签 ID，MSB 在前	
Info[4]	7～0	0～2	指令：0—读；1—写；2—擦	
Info[5]	7～0	0x40，0x80，0xC0	内部 Flash 页：0x40—保留区；0x80—用户区 1；0xC0—用户区 2	
Info[6～7]		0x0000	固定，MSB 在前	
Info[8～23]			写入指令的 16 字节内容，对读和擦无效，可以任意值填充	

保留区的容量为 256B，用户区的容量为 512B，都以记录为单位进行读/写，每条记录 16B，保留区共 16 条记录，用户区共 32 条记录。记录以追加方式写入，最后一条为有效记录。每个区写满之后，必须先擦除，才可以再写入，否则，写入失败，收不到任何回复信息。

指令不会超时，将一直有效，直到收到新的指令，或者访问成功，输出返回信息。

访问标签内存的回复：Type=0x02，Info 24B。

字　节	位	取值范围	备　注	
Type	7～0	0x02	标签回复	
Info[0～3]			标签 ID，MSB 在前	
Info[4～5]			指令返回信息	
Info[6～7]		0x0000	固定，MSB 在前	
Info[8～23]			读/写的 16B 内容	

读卡（卡号为“02000021”，内存区为“保留区”）。

发送：

FF FF FF FF FF 00 00 20 FF FF FF FE 00 00 8A 02 00 00 21 00 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 C8（40B）

返回：

FF FF FF FF FF 00 00 20（8B）
FE FE 00 06 00 00 02 02 00 00 21 00 60 00 00 FF（16B）
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF 67（16B）

最终读到的数据为：

FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

6.2.2　操作步骤

（1）参照 6.1 节的步骤建立通信连接，并进行配置、打开串口，然后进行寻卡操作。

（2）内存 Flash 页操作功能区介绍。如图 6-12 所示，功能区有三个主要功能，包括读取数据、写入数据和擦除功能；有两个下拉框，分别对应标签号选择下拉框及要操作的内存 Flash 页选择下拉框；读取到的标签数据显示在数据显示栏。

图 6-12　内存 Flash 页操作

（3）读取数据：在选定标签号后，每张标签都有三个 Flash 页选择区，分别是保留区、用户区 1 和用户区 2。三个内存区都可以进行读写操作。

下面对标签号为“02000021”保留区进行读取数据操作，读取数据成功返回如图 6-13 所示的信息。

图 6-13　读取保留区数据

再对标签号为“02000021”用户区 1 进行读取数据操作，读取数据成功返回如图 6-14 所示的信息。在读取的过程中出现了一次未选卡的错误，这是因为在读取数据前需要进行在“卡号”选择下拉框中选择卡号。

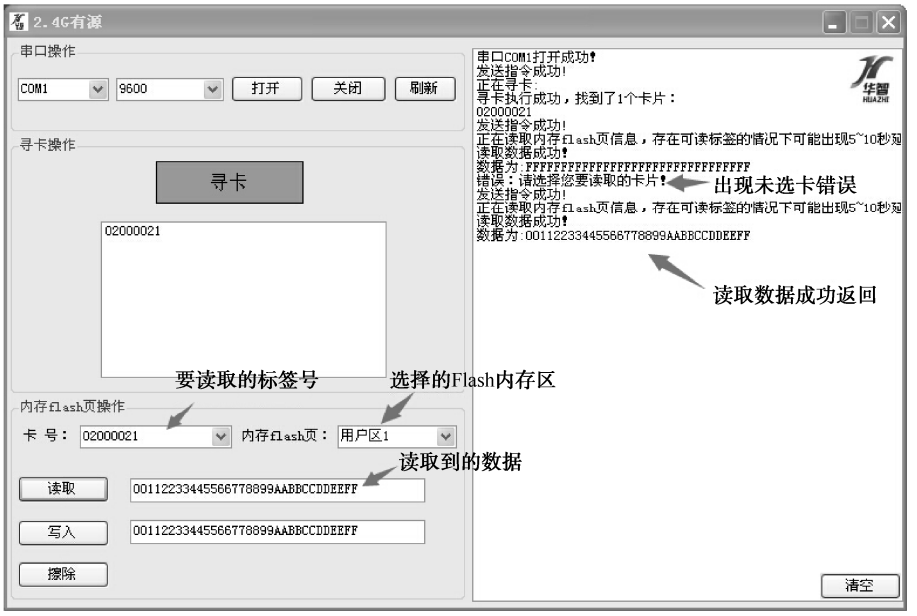


图 6-14　读取用户区 1 数据

最后对标签号为“02000021”用户区 2 进行读取数据操作，读取数据成功返回如图 6-15 所示的信息。

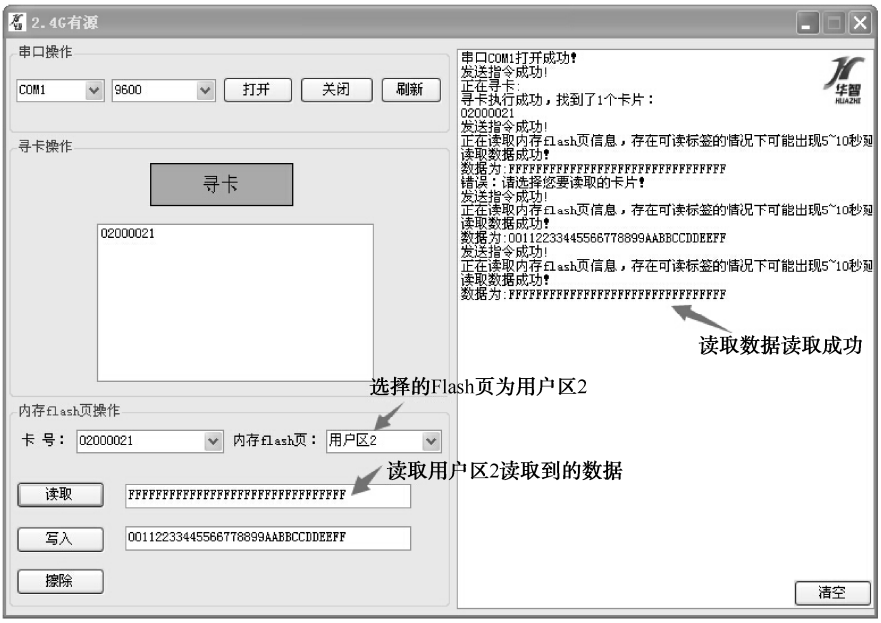


图 6-15　读取用户区 2 数据

6.2.3　程序设计：读取数据功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境下创建 C#窗体应用程序；编写代码，实现数据读取功能。主要目的是了解 Microsoft Visual Studio 2010 开发环境及窗体应用程序建立；掌握程序通过串口与 RFID 实验系统平台建立连接并通信的原理和方法；了解 RFID 实验平台 COM 协议并实现读取数据功能。

本设计所需设备如下。

（1）硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），2.4G RFID 原理模块（基于 32 位 ARM STM32 嵌入式处理器），2.4G 模块卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计相关原理：上位机应用程序采用 C#语言开发，遵守 C#编程规范，读取数据功能根据 RFID 2.4G 模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 RFID 2.4G 模块连接；上位机程序根据 RFID 2.4G 模块 COM 协议通过串口与 RFID 2.4G 模块进行通信，最终完成读取数据功能。

2．设计步骤

第一步：创建 C#窗体应用程序。

启动 Microsoft Visual Studio 2010 开发平台，创建一个如图 6-16 所示的 C#窗体应用程序，命名为“2.4G_readCard”，创建的详细方法可参考第 3 章。

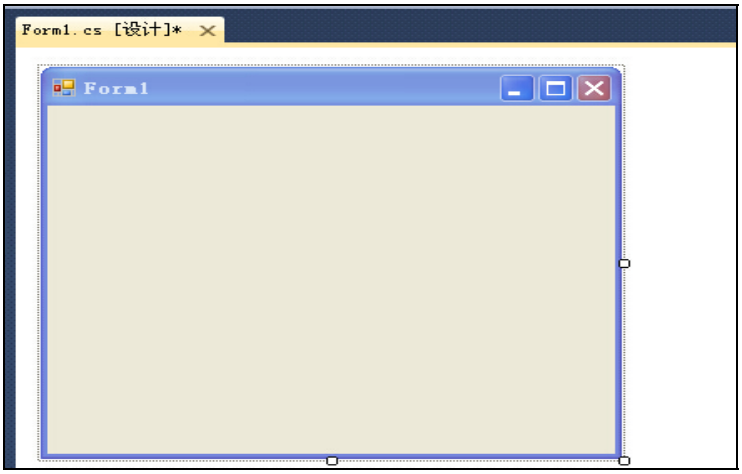


图 6-16　新建窗体

第二步：设计功能界面（前 4 步与寻卡设计部分一致）。

（1）选中新建的窗体，右击界面，在弹出的快捷菜单中选择“属性”选项，将 Text 属性改为“2.4G_readCard”。

（2）打开工具箱  ，拖出一个 GroupBox，将 Text 属性改为“串口操作”，拖出两个 ComboBox 放到串口操作的 GroupBox 中，并将其 Text 属性分别改为“串口号”与“波特率”，并将 ComboBox 与“串口号”对应的 Name 属性改为“cmb_PortNum_2p4G”，将 ComboBox 与“波特率”对应的 Name 属性改为“cmb_BaudRate_2p4G”，并将 Items 属性设置为 9600、57600、115200；拖出三个 Button 放到 GroupBox 中，分别将其 Text 属性设置为“打开”、“关闭”、“刷新”，Name 属性分别对应为“btn_Open_2p4G”、“btn_Close_2p4G”、“btn_Refresh_2p4G”。

（3）再拖出一个 GroupBox，将 Text 属性改为“寻卡操作”，将 Name 属性设置为“groupBox_Inventory”，Enabled 属性设置为 False；拖出一个 Button 放到寻卡操作的 GroupBox 中，将其 Text 属性设置为“寻卡”，将 Name 属性设置为“btn_Inventory_2p4G”；拖出一个 listBox 控件放到寻卡操作的 GroupBox 中，将 Name 属性设置为“lst_InventoryResult_2p4G”。

（4）拖出一个 ListBox 控件放在界面右侧，将其 Name 属性设置为“lsb_Info_2p4G”，拖出一个 Button 放到其右下方，将 Name 属性设置为“btn_Clear_2p4G”，Text 属性为“清空”。

（5）再拖出一个 GroupBox 放置在窗体的左下角，将 Text 属性改为“内存 flash 页操作”，将 Name 属性设置为“groupBox_flash”，Enabled 属性设置为 False；拖出两个 Label 放到内存 Flash 页操作的 GroupBox 中，将其 Text 属性分别设置为“卡 号：”、“内存 Flash 页：”；拖出两个 ComboBox 放到内存 Flash 页操作的 GroupBox 中，将其 Name 属性分别设置为“cmb_uii_2p4G”和“cmb_flash_2p4G”，将“cmb_flash_2p4G”Items 属性设置为“保留区、用户区 1、用户区 2”；拖出一个 Button 放到内存 Flash 页操作的 GroupBox 中，将其 Text 属性设置为“读取”，将 Name 属性设置为“btn_Read_2p4G”；“内存 Flash 页操作”的布局如图 6-17 所示。

最后拖出一个 TextBox 控件放到内存 Flash 页操作的 GroupBox 中，将 Name 属性设置为“txt_DataOut_2p4G”。

完成后的功能界面如图 6-18 所示。

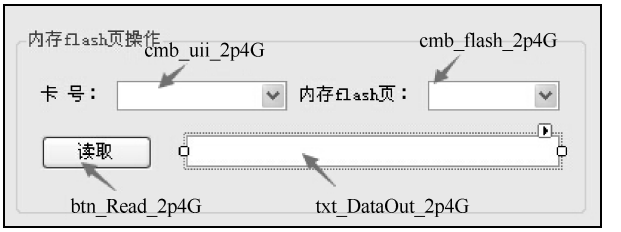


图 6-17　读取数据功能区

图 6-18　读取数据界面

第三步：编写代码实现功能（此处只附部分主要代码）。

2.4G 读取数据功能执行过程如图 6-19 所示。

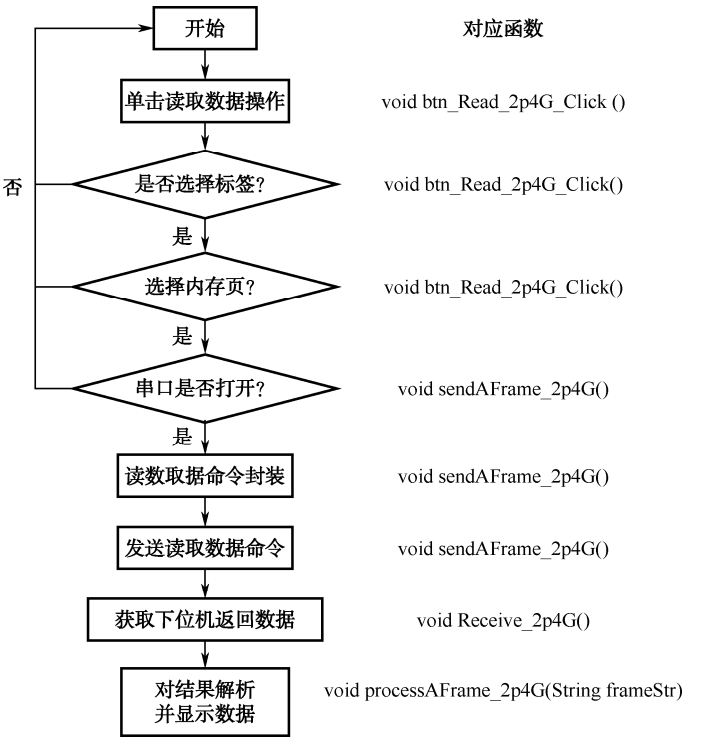


图 6-19　2.4G 读取数据功能执行过程

部分代码解析如下：

```
//读卡事件 功能：向下位机发送读卡命令
private void btn_Read_2p4G_Click(object sender, EventArgs e)
{
    if (cmb_uii_2p4G.SelectedIndex < 0)          //判断是否选择卡号
    {
        ShowInfo2p4G("错误：请选择您要读取的卡片！");
        cmb_uii_2p4G.Focus();
        return;  //如果未选择卡号，则跳出函数，不再往下执行
    }
    if (cmb_flash_2p4G.SelectedIndex < 0)        //判断是否选择了有效的 Flash 页
    {
        ShowInfo2p4G("错误：请选择您要读取的内存 Flash 页！");
        cmb_flash_2p4G.Focus();
        return;       //如果未选择有效的 Flash 页，则跳出函数，不再往下执行
    }
    CurrCMD_2p4G = READBLOCK_2P4G;     //使用变量标明当前执行读卡命令
    byte[] len = {0x00,0x20};    //命令中有效数据（除去帧头和表示长度的字节）长度为 32 字节
    byte[] addr = { 0xFF, 0xFF, 0xFF, 0xFE };    //地址
    byte type = 0x8A;                            //命令类型
    byte[] info = new byte[24];
    string strUii = cmb_uii_2p4G.Text.Trim();    //将卡号保存在字符串类型的变量中
    for (byte i = 0; i < 4; i++)
    {
        info[i] = Convert.ToByte(strUii.Substring(i * 2, 2), 16);
        //将卡号分割并进行数据类型转换保存到字节型数组中
    }
    info[4] = 0x00;                              //读取内存数据
    byte area = 0;                               //内存 Flash 页类型
    switch (cmb_flash_2p4G.SelectedIndex)        //判断选择的 Flash 页类型
    {
        case 0:                                  //Flash 页选择为保留区，则为 area 赋值 0x40
            area = 0x40;
            break;
        case 1:                                  //Flash 页选择为用户区 1，则为 area 赋值 0x80
            area = 0x80;
            break;
        case 2:                                  //Flash 页选择为用户区 2，则为 area 赋值 0xC0
            area = 0xC0;
            break;
        default:
            break;
    }
    info[5] = area;                              //将 area 保存到字节数组中
    info[6] = 0x00;
    info[7] = 0x00;
    sendAFrame_2p4G(len, addr, type, info);      //发送读卡命令
```

```
        ShowInfo2p4G("正在读取内存 Flash 页信息，存在可读标签的情况下可能出现 5～10 秒延时!");
    }

    /*其他函数声明及功能（具体函数定义见源代码）
    其他函数声明及功能参照本章其他设计部分
    */
```

第四步：编译生成程序运行。

（1）在菜单中选择“调试”→“启动调试”选项或按 F5 键生成程序并运行。

（2）选择正确的波特率和串口号，单击“打开”按钮，串口打开成功后单击“寻卡”按钮，执行寻卡操作。单击“卡号”下拉框选择读到的卡片，单击“内存 Flash 页”下拉框选择需要读取的 Flash 页，单击“读取”按钮，执行读卡操作，结果如图 6-20 所示。

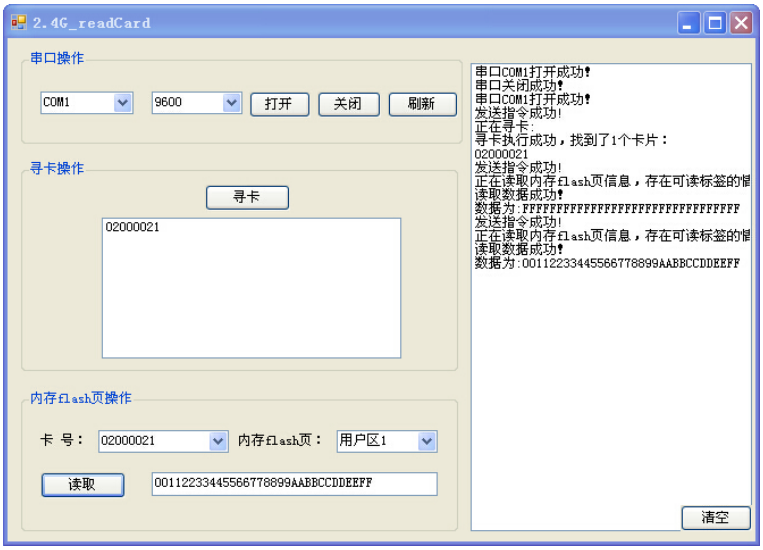


图 6-20　读取数据功能实现

6.3　2.4G 有源 RFID 写入数据操作及实现

6.3.1　协议原理

（1）通过使用上位机软件与 RFID 高频 2.4G 模块下位机进行通信，完成对 2.4G 有源标签的写入数据。

（2）典型的有源 RFID 识别系统结构组成主要包括上位机、阅读器、有源标签三大部分。上位机通过向阅读器发送指令，阅读器接收并执行来自上位机的指令，并搜索附近区域的标签，对标签进行操作，将执行后指令的结果发送给上位机。

（3）在通信时，数据的帧格式：访问标签内存与回复数据帧格式参照 6.2.1 节协议原理的内容。

【例 6-1】 写入数据（卡号为“02000021”，内存区为“保留区”），写入的数据为：

```
FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
```

此时上位机向下位机发送的数据帧：

FF FF FF FF FF 00 00 20 FF FF FF FE 00 00 8A 02 00 00 21 01 40 00 00 FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF C9 (40B)

【例 6-2】 写入数据（卡号为“02000021”，内存区为“用户区 1”），写入的数据为：

11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF

此时尚未向下位机发送的数据帧：

FF FF FF FF FF 00 00 20 FF FF FF FE 00 00 8A 02 00 00 21 01 80 00 00 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 09 (40B)

【例 6-3】 写入数据（卡号为“02000021”，内存区为：“用户区 2”），写入的数据为：

11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF

此时上位机向下位机发送的数据帧：

FF FF FF FF FF 00 00 20 FF FF FF FE 00 00 8A 02 00 00 21 01 C0 00 00 00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF 49 (40B)

6.3.2 操作步骤

（1）参照 6.1 节的步骤建立通信连接，并进行配置、打开串口，然后进行寻卡操作。

（2）内存 Flash 页操作功能区介绍。如图 6-21 所示。功能区有三个主要功能，包括读取数据、写入数据和擦除功能；有两个下拉框，分别对应标签号选择下拉框及要操作的内存 Flash 页选择下拉框，在进行写入操作时，将要写入的数据写入到对应的输入框中。

图 6-21 写入数据功能区

（3）写入数据：在选定标签号后，每张标签都有三个 Flash 页选择区，分别是保留区、用户区 1 和用户区 2。三个内存区都可以进行写入操作。

注意：写入返回时间间隔可能会稍长，而且有可能写入失败，写入数据失败大概有两种原因。第一种原因是在数据写入输入框中数据长度不足 16 字节；第二种原因是无法将要写入的数据写入 Flash 内存页中，即显示写入的数据与写入数据输入框不一致，此时则需要对存储区进行擦除操作才能继续写入。

下面对标签号为“02000021”保留区进行写入数据操作，写入数据成功返回如图 6-22 所示的信息。

当写入数据的输入框的数据长度不足 16 字节时，程序将不允许写入数据，如图 6-23 所示。

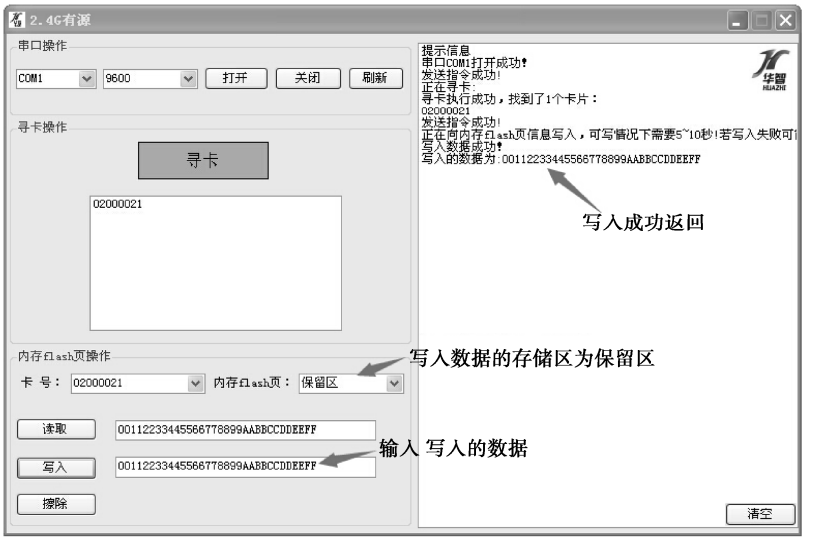


图 6-22　写入数据

图 6-23　写入数据无效

6.3.3　程序设计：写入数据功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境下创建 C#窗体应用程序；编写代码，实现写入数据功能。主要目的是了解 Microsoft Visual Studio 2010 开发环境及窗体应用程序建立；掌握程序通过串口与 RFID 实验系统平台建立连接并通信的原理和方法；了解 RFID 实验平台 COM 协议并实现写入数据功能。

本设计所需设备如下。

（1）硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），2.4G 有源 RFID 原理模块（基于 32 位 ARM STM32 嵌入式处理器），2.4G 模块卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计相关原理：上位机应用程序采用 C#语言开发，遵守 C#编程规范，写入数据功能根据 RFID 2.4G 模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 2.4G 有源 RFID 模块连接；上位机程序根据 2.4G 有源 RFID 模块 COM 协议通过串口与 2.4G 有源 RFID 模块进行通信，最终完成写入数据功能。

2．设计步骤

第一步：创建 C#窗体应用程序。

启动 Microsoft Visual Studio 2010 开发平台，创建一个如图 6-24 所示的 C#窗体应用程序，命名为“2.4G_writeCard”，创建的详细方法可参考第 3 章。

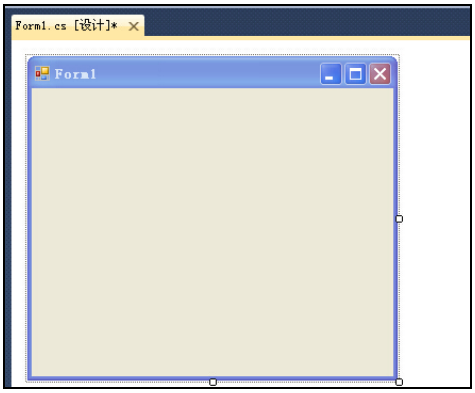


图 6-24　新建窗口

第二步：设计功能界面（前面的设计与读取数据设计部分一样）。

初步完成的界面如图 6-25 所示。

图 6-25　初步完成的界面

按读取数据设计步骤完成初步界面设计后，再拖出一个 Button 放到内存 Flash 页操作区的 GroupBox 中，位于“读取”按钮的下方，将其 Text 属性设置为“写入”，Name 属性设置为“btn_Write_2p4G”；再拖出一个 TextBox 放到内存 Flash 页操作区的 GroupBox 中，位于“写入”按钮的水平位置的右方，将 Text 属性设置为“00112233445566778899AABBCCDDEEFF”，再将 Name 属性设置为“txt_DataIn_2p4G”。

完成后的功能界面如图 6-26 所示。

图 6-26　写入数据界面

第三步：编写代码实现功能（此处只附部分主要代码）。

2.4G 写入数据功能执行过程如图 6-27 所示。

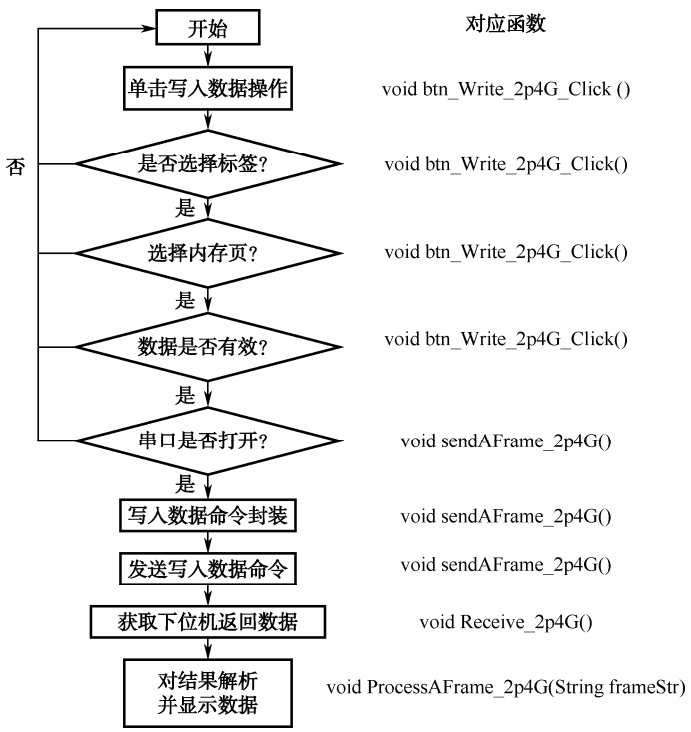


图 6-27　2.4G 写入数据功能执行过程

部分代码解析如下：

```
//写入数据事件  功能：发送写入数据命令
private void btn_Write_2p4G_Click(object sender, EventArgs e)
{
    if (cmb_uii_2p4G.SelectedIndex < 0) //判断是否选择有效的卡号
    {
        ShowInfo2p4G("错误：请选择您要写入的卡片！");
        cmb_uii_2p4G.Focus();
        return;        //未选择有效的卡号，则跳出函数不再往下执行
    }
    if (cmb_flash_2p4G.SelectedIndex < 0) //判断是否选择要操作的内存 Flash 页
    {
        ShowInfo2p4G("错误：请选择您要写入的内存 Flash 页！");
        cmb_flash_2p4G.Focus();
        return;        //为选择要操作的内存 Flash 页，跳出函数不再往下执行
    }
    byte[] len={0x00,0x20};      //命令中有效数据（除去帧头和表示长度的字节）长度为 32 字节
    byte[] addr = { 0xFF, 0xFF, 0xFF, 0xFE }; //地址
    byte type = 0x8A;        //命令类型
    byte[] info=new byte[24];  //字节型数组，保存操作类型（读取数据或写入数据）和写入的数据
    string strUii = cmb_uii_2p4G.Text.Trim(); //将卡号保存到字符串变量中
    for (byte i = 0; i < 4; i++)
    {                                    //将卡号分割并进行数据类型转换保存到字节型数组中
        info[i] = Convert.ToByte(strUii.Substring(i * 2, 2), 16);
    }
    info[4] = 0x01;                      //命令为写入内存数据
    byte area = 0;                       //内存 Flash 页类型
    switch (cmb_flash_2p4G.SelectedIndex)
    {
        case 0:                          //Flash 页选择为保留区，则为 area 赋值 0x40
            area = 0x40;
            break;
        case 1:                          //Flash 页选择为用户区 1，则为 area 赋值 0x80
            area = 0x80;
            break;
        case 2:                          //Flash 页选择为用户区 2，则为 area 赋值 0xC0
            area = 0xC0;
            break;
        default:
            break;
    }
    info[5] = area;                      //将 area 保存到字节数组中
    info[6] = 0x00;
    info[7] = 0x00;
    String str_data = txt_DataIn_2p4G.Text.Trim(); //将写入的数据保存到字符串变量中
    //Byte[] data_Write = new Byte[16];
    try
```

```
        {
            for (byte i = 0; i < 16; i++)
            {   //将写入的数据分割并进行数据类型转换保存到字节型数组中
                info[8 + i] = (byte)(Convert.ToByte(str_data.Substring(i * 2, 2), 16));
            }
        }
        catch                                   //填写无效的数据
        {
            ShowInfo2p4G("您填写的数据无效！请输入 16 字节的十六进制有效数据");
            txt_DataIn_2p4G.Focus();
            return;
        }
        CurrCMD_2p4G = WRITEBLOCK_2P4G; // CurrCMD_2p4G 变量标明当前执行写入数据命令
        sendAFrame_2p4G(len, addr, type, info); //发送写入数据命令
        ShowInfo2p4G("正在向内存 flash 页信息写入，可写情况下需要 5～10 秒!\n 若写入失败可能是
                    该区已有信息，请尝试先擦除后再写入!");
    }
    /*其他函数声明及功能（具体函数定义见源代码）
    其他函数声明及功能参见本章其他设计部分
    */
```

第四步：编译生成程序运行。

（1）在菜单中选择“调试”→“启动调试”选项或按 F5 键生成程序并运行。

（2）选择正确的波特率和串口号，单击“打开”按钮，串口打开成功后执行寻卡操作。单击“卡号”下拉框选择读到的卡片，单击“内存 Flash 页”下拉框选择需要写入的 Flash 页，单击“写入”按钮，执行写入操作，结果如图 6-28 所示。

图 6-28　写入数据实现

6.4　2.4G 有源 RFID 擦除数据操作及实现

6.4.1　协议原理

（1）通过使用上位机软件与 RFID 高频 2.4G 模块下位机进行通信，完成对 2.4G 有源标签的擦除数据操作。

（2）典型的有源 RFID 识别系统结构组成主要包括上位机、阅读器、有源标签三大部分。上位机通过向阅读器发送指令，阅读器接收并执行来自上位机的指令，并搜索附近区域的标签，对标签进行操作，将执行后指令的结果发送给上位机。

在通信时，数据的帧格式：访问标签内存与回复数据帧格式参照 6.2.1 节协议原理的内容。

擦除数据（擦除的卡号为“02000021”，擦除的内存 Flash 页为“保留区”）

此时上位机向下位机发送：

```
FF FF FF FF FF 00 00 20 FF FF FF FE 00 00 8A 02 00 00 21 02 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 CA (40B)
```

6.4.2　操作步骤

（1）参照 6.1 节的步骤建立通信连接，并进行配置、打开串口，然后进行寻卡操作。

（2）擦除数据操作，擦除成功返回如图 6-29 所示的信息。

擦除数据后，初始化数据为：

```
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

图 6-29　擦除数据

6.4.3 程序设计：擦除数据功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境下创建 C#窗体应用程序；编写代码，实现擦除数据功能。主要目的是了解 Microsoft Visual Studio 2010 开发环境及窗体应用程序建立；掌握程序通过串口与 RFID 实验系统平台建立连接并通信的原理和方法；了解 RFID 实验平台 COM 协议并实现擦除数据功能。

本设计所需设备如下。

（1）硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），2.4G RFID 原理模块（基于 32 位 ARM STM32 嵌入式处理器），2.4G 模块卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计主要原理：上位机应用程序采用 C#语言开发，遵守 C#编程规范，擦除数据功能根据 RFID 2.4G 模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 RFID 2.4G 模块连接；上位机程序根据 RFID 2.4G 模块 COM 协议通过串口与 RFID 2.4G 模块进行通信，最终完成擦除数据的功能。

2．设计步骤

第一步：创建 C#窗体应用程序。

启动 Microsoft Visual Studio 2010 开发平台，创建如图 6-30 所示的 C#窗体应用程序，命名为“2.4G_eraseCard”，创建的详细方法可参考第 3 章。

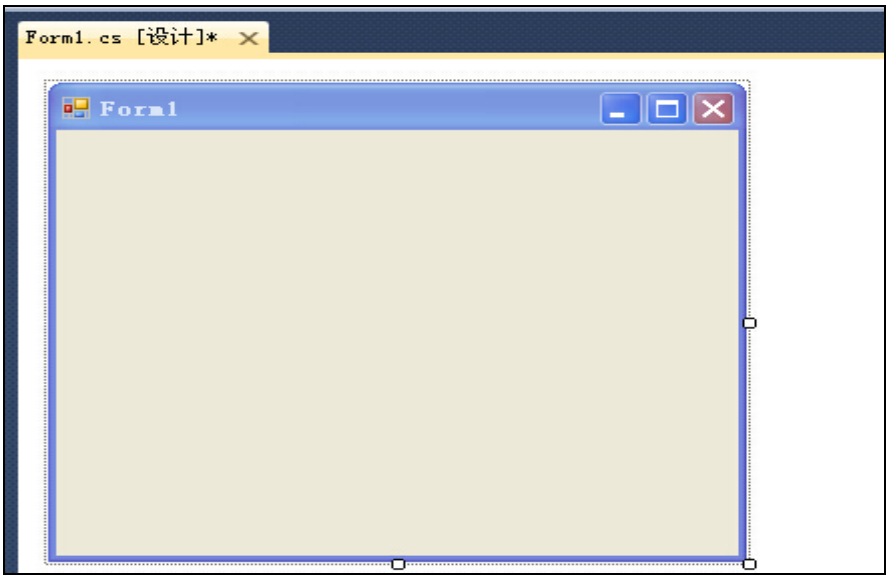


图 6-30　新建窗体

第二步：设计功能界面。

（1）选中新建的窗体，右击界面，在弹出的快捷菜单中选择“属性”选项，将 Text 属性改为“2.4G_eraseCard”。

（2）打开工具箱  ，拖出一个 GroupBox，将 Text 属性改为“串口操作”，拖出两个 ComboBox 放到串口操作的 GroupBox 中，并将其 Text 属性分别改为“串口号”与“波特率”，并将 ComboBox 与“串口号”对应的 Name 属性改为“cmb_PortNum_2p4G”，将 ComboBox 与“波特率”对应的 Name 属性改为“cmb_BaudRate_2p4G”，并将 Items 属性设置为 9600、57600、115200；拖出三个 Button 放到 GroupBox 中，分别将其 Text 属性设置为“打开”、“关闭”、“刷新”，Name 属性分别对应为“btn_Open_2p4G”、“btn_Close_2p4G”、“btn_Refresh_2p4G”。

（3）再拖出一个 GroupBox，将 Text 属性改为“寻卡操作”，将 Name 属性设置为“groupBox_Inventory”，Enabled 属性设置为 False；拖出一个 Button 放到寻卡操作的 GroupBox 中，将其 Text 属性设置为“寻卡”，将 Name 属性设置为“btn_Inventory_2p4G”；拖出一个 listBox 控件放到寻卡操作的 GroupBox 中，将 Name 属性设置为“lst_InventoryResult_2p4G”。

（4）拖出一个 ListBox 控件放在界面右侧，将其 Name 属性设置为“lsb_Info_2p4G”，拖出一个 Button 放到其右下方，将 Name 属性设置为“btn_Clear_2p4G”，Text 属性为“清空”。

（5）再拖出一个 GroupBox 放置在窗体的左下角，将 Text 属性改为“内存 Flash 页擦除操作”，将 Name 属性设置为“groupBox_flash”，Enabled 属性设置为 False；拖出两个 Label 放到内存 Flash 页擦除操作的 GroupBox 中，将其 Text 属性分别设置为“卡 号：”、“内存 Flash 页：”；拖出两个 ComboBox 放到内存 Flash 页擦除操作的 GroupBox 中，将其 Name 属性分别设置为“cmb_uii_2p4G”和“cmb_flash_2p4G”，将“cmb_flash_2p4G”的 Items 属性设置为“保留区、用户区 1、用户区 2”；最后拖出一个 Button 放到内存 Flash 页擦除操作的 GroupBox 中，将其 Text 属性设置为“擦除”，将 Name 属性设置为“btn_Erase_2p4G”。

完成这五步操作的界面如图 6-31 所示。

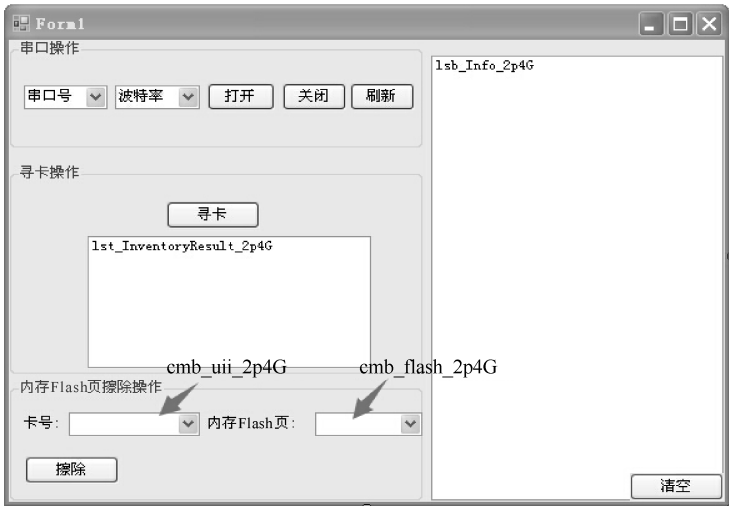


图 6-31　擦除功能界面

第三步：编写代码实现功能（此处只附部分主要代码）。

2.4G 擦除数据功能执行过程如图 6-32 所示。

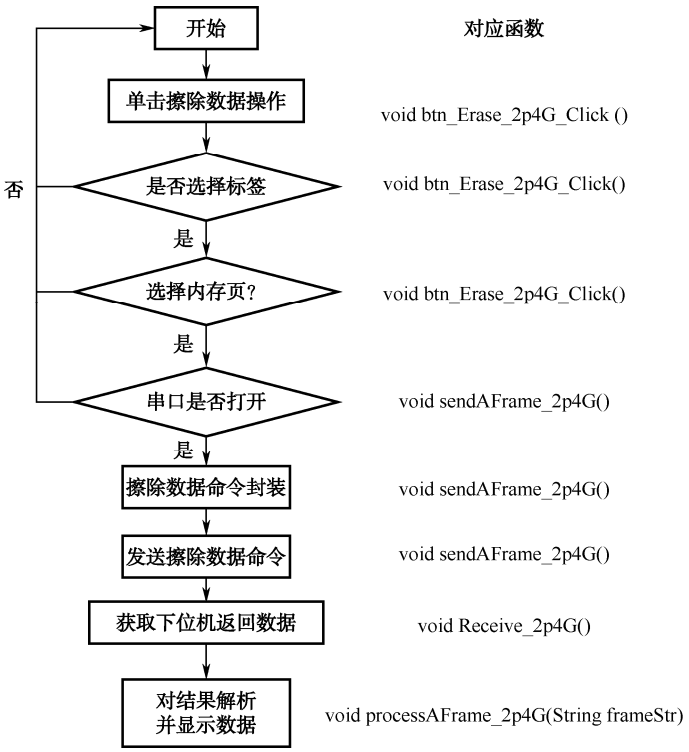


图 6-32　2.4G 擦除数据功能执行过程

部分代码解析如下：

```
//擦除事件 功能：向下位机发送擦除数据命令
private void btn_Erase_2p4G_Click(object sender, EventArgs e)
{
    if (cmb_uii_2p4G.SelectedIndex < 0)     //判断是否选择了有效的卡号
    {
        ShowInfo2p4G("错误：请选择您要擦除的卡片！");
        cmb_uii_2p4G.Focus();
        return;                             //为选择有效卡号，跳出函数不再往下执行
    }
    if (cmb_flash_2p4G.SelectedIndex < 0)  //判断是否选择有效的内存 Flash 页
    {
        ShowInfo2p4G("错误：请选择您要擦除的内存 Flash 页！");
        cmb_flash_2p4G.Focus();
        return;                             //未选择有效的 Flash 页，跳出函数不再往下执行
    }
    CurrCMD_2p4G = ERASEBLOCK_2P4G; //CurrCMD_2p4G 变量标明当前执行擦除命令
    byte[] len={0x00,0x20};   //命令中有效数据（除去帧头和表示长度的字节）长度为 32 字节
    byte[] addr = { 0xFF, 0xFF, 0xFF, 0xFE }; //地址
    byte type = 0x8A;                       //命令类型
    byte[] info = new byte[24];
    string strUii = cmb_uii_2p4G.Text.Trim(); //将卡号保存到字符串型变量中
    for (byte i = 0; i < 4; i++)
    {                                       //将卡号分割并进行数据类型转换保存到
```

```
            info[i] = Convert.ToByte(strUii.Substring(i * 2, 2), 16);
        }
        info[4] = 0x02;                                  //命令为擦除内存数据
        byte area = 0;                                   //内存 Flash 页类型
        switch (cmb_flash_2p4G.SelectedIndex)
        {
            case 0:                                      //Flash 页选择为保留区，则为 area 赋值 0x40
                area = 0x40;
                break;
            case 1:                                      //Flash 页选择为用户区 1，则为 area 赋值 0x80
                area = 0x80;
                break;
            case 2:                                      //Flash 页选择为用户区 2，则为 area 赋值 0xC0
                area = 0xC0;
                break;
            default:
                break;
        }
        info[5] = area;                                  //将 area 保存到字节数组中
        info[6] = 0x00;
        info[7] = 0x00;
        sendAFrame_2p4G(len, addr, type, info);          //发送擦除数据命令
        ShowInfo2p4G("正在擦除内存 Flash 页信息，存在可擦除标签的情况下可能出现 5～10 秒延时!");
    }
    /*其他函数声明及功能（具体函数定义见源代码）
    其他函数声明及功能参照本章其他设计部分*/
```

第四步：编译生成程序运行。

（1）在菜单中选择“调试”→“启动调试”选项或按 F5 键生成程序并运行。

（2）选择正确的波特率和串口号，单击“打开”按钮，串口打开成功后单击“寻卡”按钮，执行寻卡操作。单击“卡号”下拉框选择读到的卡片，单击“内存 Flash 页”下拉框选择需要擦除的 Flash 页，单击“擦除”按钮，执行擦除操作，结果如图 6-33 所示。

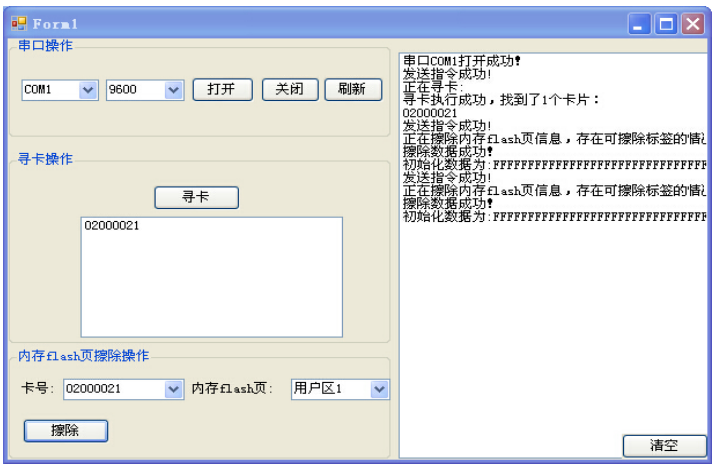


图 6-33　擦除功能实现

第 7 章　RFID 低频 125K ID 卡协议原理及实践开发

本章内容设备包括桂林华智 RFID 物联网教学科研平台实验箱 125K 模块读写器、125K 模块标签、串口线。主要内容为通过使用上位机软件，与 RFID 125K 模块下位机进行通信，完成对 125K 模块标签的操作。通过本章的学习，掌握对 125K 模块读写器及卡的操作，了解相关协议及工作原理。

（1）TX125 系列非接触 IC 卡射频读卡模块采用 125K 射频基站。当有卡靠近模块时，模块会根据韦根协议（Wiegand 协议是国际上统一的标准，是由摩托罗拉公司制定的一种通信协议。它适用于涉及门禁控制系统的读卡器和卡片的许多特性。它有很多格式，标准的 26 位应该是最常用的格式，此外还有 34 位、37 位等格式）或 UART（本文通信方式为串口方式）方式输出 ID 卡卡号，用户仅需简单地进行读取操作即可，在串口方式下，可以在主动与被动模式下工作。该读卡模块完全支持 EM 及其兼容卡片的操作，非常适合于门禁、考勤等系统的应用。

（2）模块平面图，如图 7-1 所示。

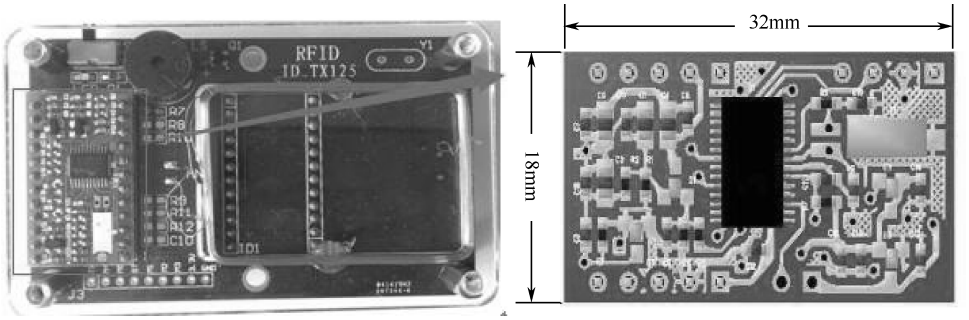
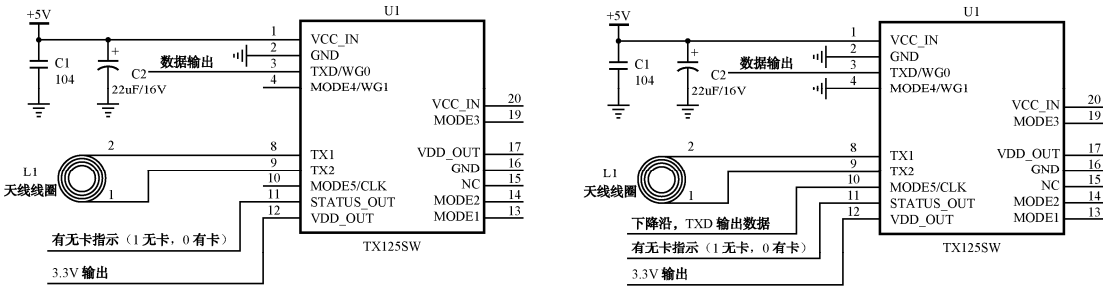


图 7-1　模块平面图

（3）产品原理图（本章主要讲述主动模式下操作），如图 7-2 所示。



（a）主动串口模式下　　（b）被动串口模式下

图 7-2　产品原理图

（4）TX125 模块的特点：体积小巧、简单、易用、性价比高；支持 EM 及其兼用卡；可选择低功耗模式；读卡距离远（30～100mm）;模块内部具有看门狗，永不死机；工作范围宽，低温可到-40℃。

7.1 协议原理

所谓通信协议，就是读卡模块以何种格式把读取到的卡号发送出来。TX125 支持韦根接口和串口两种协议（下文内容选择为串口通信）。

UART 接口一帧的数据格式为：1 个起始位、8 个数据位、无奇偶校验位、1 个停止位。波特率可选择 9600bps 或者 19200bps。

下位机向上位机发送数据帧格式：6 字节数据，高位在前，格式为 5 字节数据+1 字节校验位（CRC 检验）。例如，卡号数据为 0B00D5F0C7，则输出为 0x0B 0x00 0xD5 0xF0 0xC7 0xE9（校验和计算：0x0B^0x00^0xD5^0xF0^0xC7=0xE9）。其中，在卡号数据中，第一字节“0x0B”一般是厂家码。中间 4 个字节“0x00 0xD5 0xF0 0xC7”是卡片的序号。

在串口方式下，可以在主动与被动模式下工作。

主动模式：当有卡进入该射频区域内时，主动发出以上格式的卡号数据。

被动模式：CLK 的下降沿触发卡号的输出，格式为以上数据格式。操作方法为：在准备读取卡号之前，打开串口中断并启动超时定时器（80ms），将一直保持高电平的 CLK 置低电平，产生下降沿并一直保持低电平，等待卡号数据接收，若接收到卡号存储待用，若在等待过程中无数据接收，且超时定时器已经溢出，则标识本次读取卡号失败；无论成功与失败最后都将 CLK 重置高电平，进入待机，以便下一次读取卡号。

由于工作模式为主动模式，下位机获取到数据就会主动向上位机发送。因此上位机若要获取读写器读取到的数据只需打开通信连接并打开接收数据线程（接收线程读取并解析串口数据）。

当有标签靠近读写区时,下位机往上位机发送的指令为：

```
01 00 AE 38 73 E4 (6 字节,前 5 个字节为标签号,第 6 个字节 E4 为 CRC 校验位)
```

7.2 操作步骤

（1）125K 操作界面功能介绍（通信协议为串口通信，工作模式为主动模式）。

如图 7-3 所示，串口操作包括打开、关闭和刷新三个按钮，波特率默认为 9600。操作区主要有两个功能按钮（在串口未打开的情况下处于不可单击状态），分别是“重启接收线程”和“终止接收线程”。“重启接收线程”功能是为接收数据线程初始化新的实例并启动接收数据线程；“终止接收线程”功能是终止接收数据线程。卡号数据显示区域显示内容包括类型（R 表示接收数据）、时间（获取数据的时间）、卡号（获取到的卡号数据）及 CRC（CRC 校验）。

（2）打开串口对数据进行获取。125K 串口打开功能如图 7-4 所示。

由于当前 125K 读写器工作模式为主动模式，有卡进入该射频区域内时，读写器主动发出卡号数据。因此，若要接收卡号数据，需要打开串口建立通信连接。成功打开串口（打开串口操作包含初始化并启动线程功能）后，对数据进行获取。获取数据成功返回如图 7-5 所示的信息。

图 7-3　界面设计

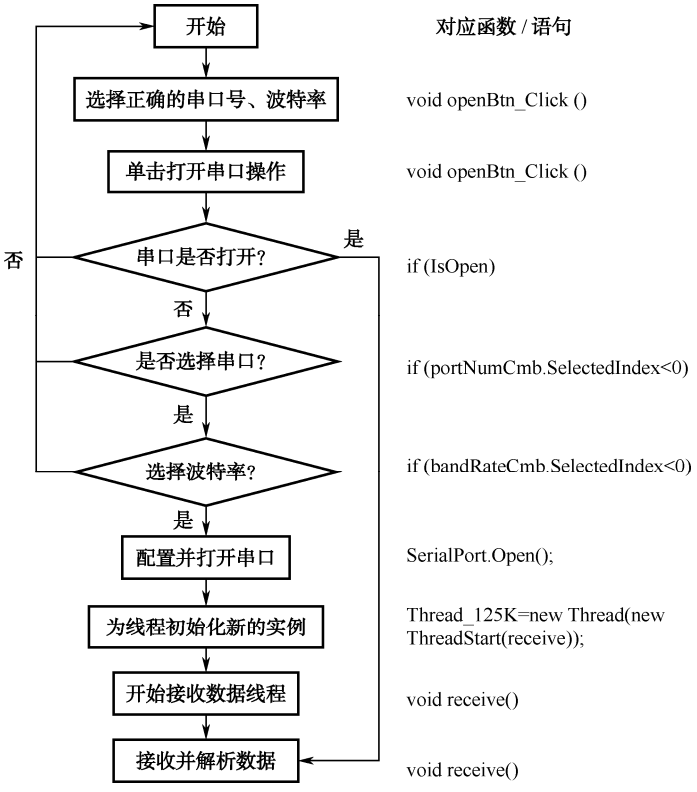


图 7-4　125K 串口打开功能

图 7-5　打开串口

此时，单击“终止接收线程”按钮，接收数据线程就会终止。读写器当前工作在主动模式下，当卡片进入该射频区域内时，读写器就会获取到标签的卡号数据并向串口发送数据，但是由于接收上位机接收线程终止，因此上位机无法获取串口数据，如图 7-6 所示。此时若需要重新获取数据，需要单击“重启接收线程”按钮重新初始化并启动接收数据线程。

图 7-6　终止接收线程

可能导致寻卡异常的原因：读写器一次只能识别一张标签卡，当多张标签卡进入射频区域内时，将导致读写无法识别标签。

7.3　程序设计：寻卡、数据显示功能的实现

1．设计内容

在 Microsoft Visual Studio 2010 开发环境下创建 C#窗体应用程序；编写代码，实现寻卡功

能并将数据显示出来。主要目的是掌握程序通过串口与 RFID 实验系统平台建立连接并通信的原理和方法；了解 RFID 实验平台 COM 协议并实现寻卡功能；了解 RFID 实验平台 COM 协议并实现数据显示功能。

本设计所需设备如下。

（1）硬件：PC（Pentium 500 以上，硬盘 80GB 以上，内存大于 1GB，Windows 操作系统），125K（低频 125kHz）RFID 模块（基于 8 位 ARM STM32 嵌入式处理器），125K 卡片，串口线，USB 转串口线。

（2）软件：Microsoft Visual Studio 2010。

本设计主要原理：上位机应用程序采用 C#语言开发，遵守 C#编程规范，寻卡功能根据 RFID 低频模块 COM 协议实现，使用串口线和 USB 转串口线将 PC 与 RFID 低频模块连接；上位机程序根据 RFID 低频模块 COM 协议通过串口与 RFID 低频模块进行通信，最终完成寻卡、显示卡的数据功能。

2．设计步骤

第一步：创建 C#窗体应用程序。

启动 Microsoft Visual Studio 2010 开发平台，创建一个如图 7-7 所示的 C#窗体应用程序，命名为“125K”，创建的详细方法可参考第 3 章。

图 7-7　新建窗体

第二步：设计功能界面。

（1）选中新建的窗体，右击界面，在弹出的快捷菜单中选择“属性”选项，将 Text 属性改为“125K”。

（2）打开工具箱  ，拖出一个 GroupBox，将 Text 属性改为“串口操作”，拖出两个 Label 放到串口操作的 GroupBox 中，并将其 Text 属性分别改为“串口号”与“波特率”，拖出两个 ComboBox 放到串口操作的 GroupBox 中，并将 ComboBox 与“串口号”对应的 Name 属性改为“portNumCmb”，将 ComboBox 与“波特率”对应的 Name 属性改为“bandRateCmb”，并将 Items 属性设置为 7200、9600、115200，将 Text 属性设置为“9600”；拖出三个 Button 放到 GroupBox 中，分别将其 Text 属性设置为“打开”、“关闭”、“刷新”，

Name 属性分别对应为“openBtn”、“closeBtn”、“refreshBtn”。

（3）拖出一个 GroupBox ，将其 Text 属性改为 “操作”，拖出两个 Button 放入其中，分别将 Button 的 Text 属性命名为“重启接收线程”、“终止接收线程”，将对应的 Name 属性改为“openANTBtn”、“closeANTBtn”。

（4）拖出一个 ListBox 控件，将其 Name 属性设置为“listInfo”，拖出一个 Button 放到其中，将 Name 属性设置为“clearListInfoBtn”，Text 属性为“清空”。

（5）拖出一个 dataGridView，将其 Name 属性设置为“dataGridView1”，选择其 Items 属性，单击“打开”按钮，出现如图 7-8 所示的对话框，然后单击“添加”按钮，弹出如图 7-9 所示的对话框。在“名称”文本框填写列的属性名“Type”，把页眉文本写为“类型”（名称即是 Name 的属性，页眉文本则为页面表头显示的文字）。以同样的方式，添加行“时间”、“卡号”、“CRC”，对应的 Name 属性为“time2”、“CardNum”、“CRC”。添加成功后返回到图 7-8 中调整 Width 属性。在 DataGridView 的右下角添加一个 Button 按钮，将其 Text 属性改为“清空”，Name 属性为“clearRowBtn”。

最后设计好的界面如 7-10 所示。

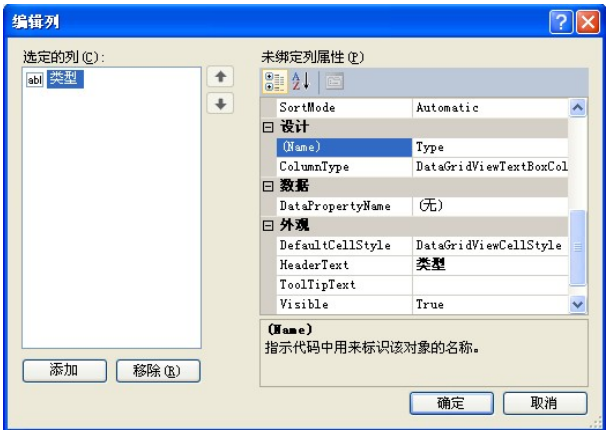


图 7-8　添加列

图 7-9　列命名

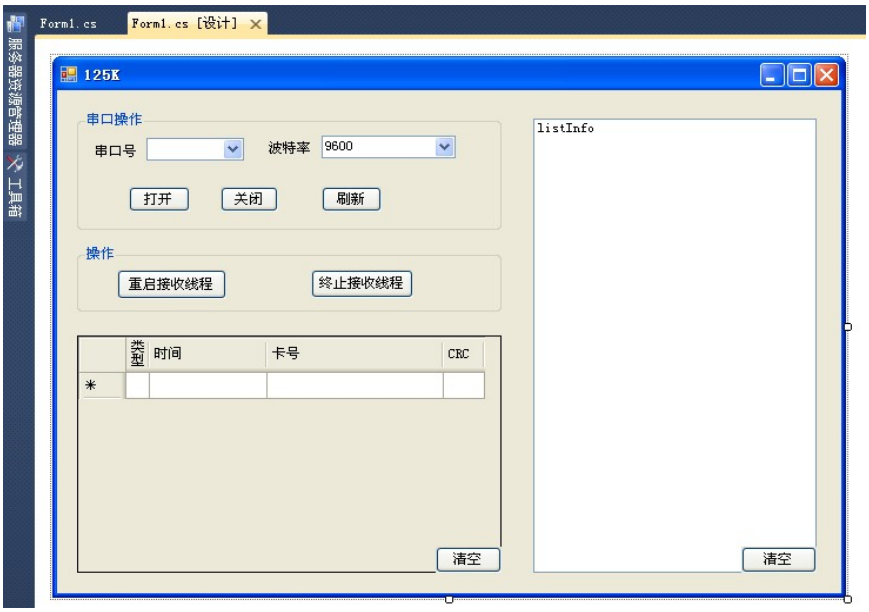


图 7-10　完成界面

第三步：编写代码实现功能（此处只附部分主要代码及解析）。

125K 打开接收线程功能如图 7-11 所示。

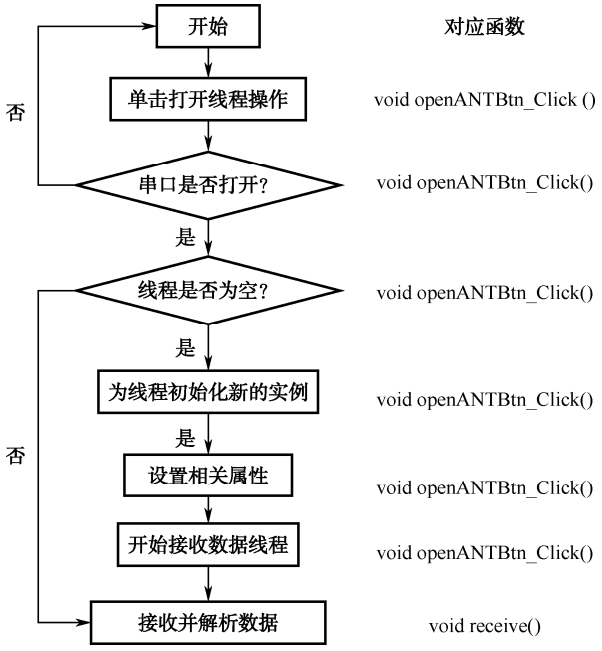


图 7-11　125K 打开接收线程功能

部分代码解析如下：

```
//接收数据函数线程
private void receive()
{
    Byte[] R_Data = new Byte[6];      //字节型数组,用于保存接收的数据
```

```
while (IsOpen)//判断串口是否打开
{
    try
    {
        if (UseSerialPort)//判断是否使用串口通信
        {
            if (SerialPort.BytesToRead > 0) //判断是否从串口中读取到数据
            {
                Int32 count = SerialPort.Read(R_Data, 0, R_Data.Length);
                //读取串口数据,并返回数据长度
                String str = ""; //新建变量,用于保存串口读取到的数据
                for (Int32 i = 0; i < count; i++)
                {
                    str = str + String.Format("{0:X2}",R_Data[i]);//保存串口数据到 str 中
                }
                String time = System.DateTime.Now.ToLongTimeString();
                //获取系统当前时间
                if (count < 6) //判断天线打开且接收到的数据不足 6 个字节
                {
                    Thread.Sleep(50);//接收数据线程睡眠 50ms
                    Int32 n = SerialPort.Read(R_Data, 0, R_Data.Length);//读取串口数据
                    count += n; //将两次的长度进行相加
                    for (Int32 i = 0; i < n; i++)
                    {
                        str = str + String.Format("{0:X2}", R_Data[i]); //字符串合并
                    }
                }
                AddListInfo(String.Format("{0}:收到{1}个字节。", time, count));//信息显示
                if (count == 6)
                {
                    addRow(str); //显示接收数据
                    Int32 sum = R_Data[0] + R_Data[1] + R_Data[2] + R_Data[3] +
                                R_Data[4] + R_Data[5];
                    if (sum == 0) //判断收到数据的和是否等于 0
                    {
                        throw (new Exception("收到全 0 的数据，请检查设备连接！"));
                    }
                }
                Thread.Sleep(50); //接收数据线程睡眠 50ms
            }
        }
        else
        {}
    }
    catch (Exception ex)
    { AddListInfo(String.Format("错误：数据接收异常，异常信息为{0}。", ex.Message));
        IsOpen = false;break;}
        Thread.Sleep(50); //接收数据线程睡眠 50ms
```

```
            }
            try
            {
                if (UseSerialPort)                                  //使用串口通信下
                {
                    SerialPort.Close();                             //关闭串口
                    SerialPort = null;                              //使串口引用为空
                    AddListInfo(String.Format("串口关闭成功"));  //显示提示信息
                }
                else                                                //使用网口通信下
                {
                    socket.Shutdown(SocketShutdown.Both);           //网口禁用
                    socket.Close();   //关闭网口通信
                    socket = null;    //网口引用为空
                    AddListInfo(String.Format("网络连接断开成功！"));
                }
            }
            catch { }
        }
```

第四步：编译生成程序运行。

将 125K 低频模块通过 USB 数据线与 PC 进行连接。编译运行程序，选择正确的串口和波特率，然后单击“打开”按钮，将 125K 标签卡放置于天线上方，此时读取到卡片数据，如图 7-12 所示。

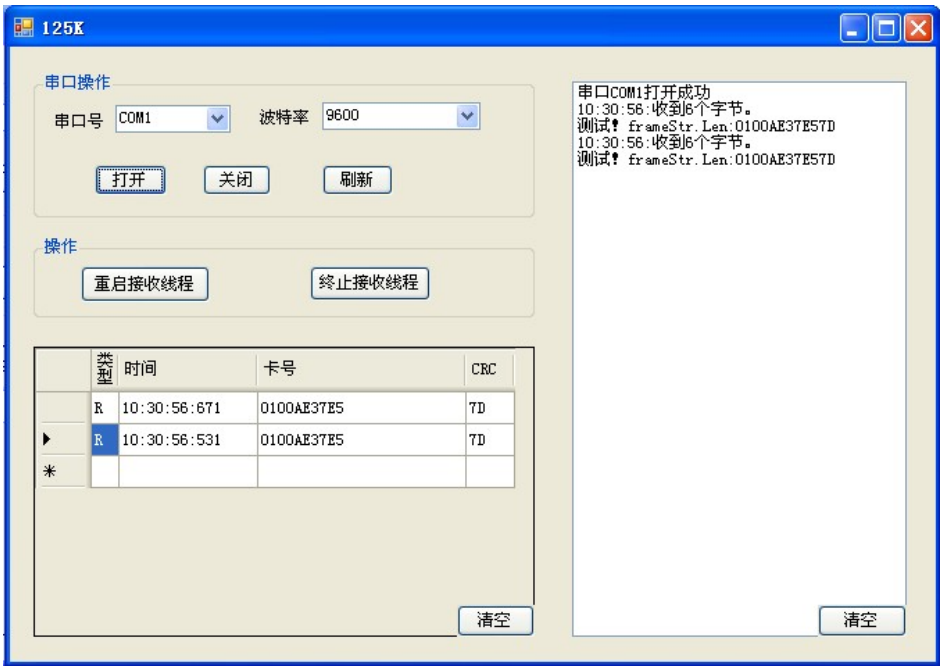


图 7-12　寻卡实现

第 8 章　RFID 与 ZigBee 互联原理及实践开发

在本章中，RFID 模块将作为“传感器”的角色搭载在 ZigBee 节点上，RFID 负责采集数据，并将数据传给 ZigBee 节点。数据经过无线短距离传输传给协调器，协调器将数据经过串口发给上位机。本章内容所使用的设备有 ZigBee 模块、RFID 实验平台、上位机、串口线。通过本章的学习，掌握 RFID 与 ZigBee 互联的实验操作，理解数据传输过程，了解相关协议及原理。

ZigBee 是一种标准，该标准定义了短距离、低数据传输速率无线通信所需要的一系列通信协议。基于 ZigBee 的无线网络所使用的工作频段为 868MHz、915MHz 和 2.4GHz，最大数据传输速率为 250kbps。

8.1　RFID 与 ZigBee 互联原理

数据传输过程如图 8-1 所示。

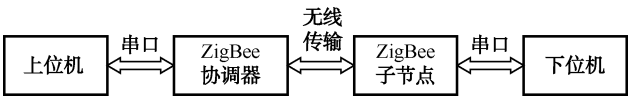


图 8-1　数据传输过程

上位机与 RFID 各个读卡器之间是通过串口进行通信的。串口的传输距离有限，并且在实际应用中有可能会需要无线网络连接读卡器的情况。因此这里引入了无线传感器网络 ZigBee 协议。ZigBee 协议具有自组网功能，并且协调器可方便地和网络内的所有子节点进行通信，所以上位机只需通过串口连接一个 ZigBee 协调器，RFID 读卡器连接一个 ZigBee 子节点，上位机就可以方便地对该读卡器进行无线操作了。其各模块连接示意图如图 8-2 所示。

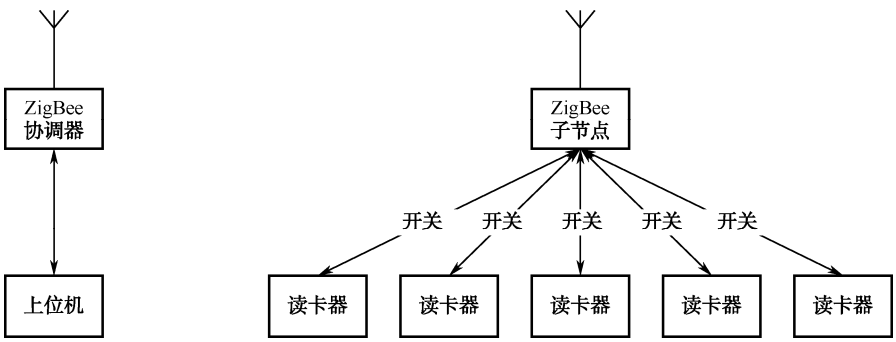


图 8-2　各模块连接示意图

8.2　ZigBee 与 RFID ISO 15693 模块互联实践开发

1．开发流程

下面以 ZigBee 与 RFID ISO 15693 模块互联为例，实现 HF13.56MHz ISO 15693 数据块的读/写功能，其他 RFID 模块与 ZigBee 互联参照此例。这里所使用设备有桂林华智 CC2530 ZigBee 模块、RFID 物联网教学科研平台实验箱高频 15693 模块读写器。

（1）组建 ZigBee 网络

打开实验箱开关，给 ZigBee 协调器和子节点上电，待节点上 LED1 灯都亮起后，证明 ZigBee

网络已经组建好了。

（2）模块选通

把计算机串口与 ZigBee 协调器串口相连接；通过“自锁”开关把相关读卡器的串口与 ZigBee 子节点串口相连接，其他读卡器则断开串口连接，如需做 ISO 15693 读卡器实验，则闭合“SS0”开关，其他“SS1”、“SS2”、“SS3”、“SS4”开关断开。

（3）配置 ZigBee 模块波特率

通过协调器上面的 K1 按键，选择相应的波特率，如 ISO 15693 读卡器串口输出的波特率为 115200kbps，则需通过协调器 K1 按键进行调节，使协调器和子节点两边的波特率都设置为 115200kbps。

完成上述三个步骤后即完成了 ZigBee 和 RFID 模块的配置，接下来即可打开上位机软件进行数据块读/写。

（4）打开 HF13.56MHz ISO 15693 演示部分上位机软件进行数据块的读写操作。

打开 HF13.56MHz ISO 15693 演示部分上位机软件并选择正确的串口号，波特率选择 115200kbps，打开串口后即可进行操作。

2．上位机软件实现数据块读写操作

（1）打开上位机软件，再启动 RFID 原理机的电源，建立通信连接。

（2）读取卡片的 UID。进行数据读取操作，首先需要进行寻卡操作。将卡片放到识别区（如果连接了外接天线，将卡片放置在天线旁边。这里所有实验均采用外接天线为例），选择“寻卡”命令，再选中“单卡识别”单选按钮，然后单击“寻卡”按钮后即执行“寻卡”操作，寻卡成功返回如图 8-3 所示的信息。

图 8-3　寻卡操作

（3）数据块数据读取。选择读取数据块、卡号、填写地址、长度、数量（系统已经默认，可以不改写）。单击“读取多个数据块”按钮即可读取数据，数据读取成功后的信息如图 8-4

所示，地址“0”表示读取第 0 块，数量“2”则表示读取 2 块数据，读取到的数据为“12345678 87654321”，表示 0 块的 8 个字节。

图 8-4　数据块读取

（4）获取卡片的信息，读取到卡的块数和每个块的字节数，如应用族（AFI）、数据存储标识 DSFID 等。选择“获取系统信息”命令，单击“获取系统信息”按钮即可执行“获取系统信息”操作，成功返回如图 8-5 所示的信息，其中“应用族标识”为“00”，“数据存储格式标识”为“00”，“数据块数量”为“1B”，由于块的编号是从 0～1B 开始的，因此共有数据模块 1B+1 个，同理每个数据模块的长度也是 3+1 个，即 4B。

图 8-5　获取系统信息

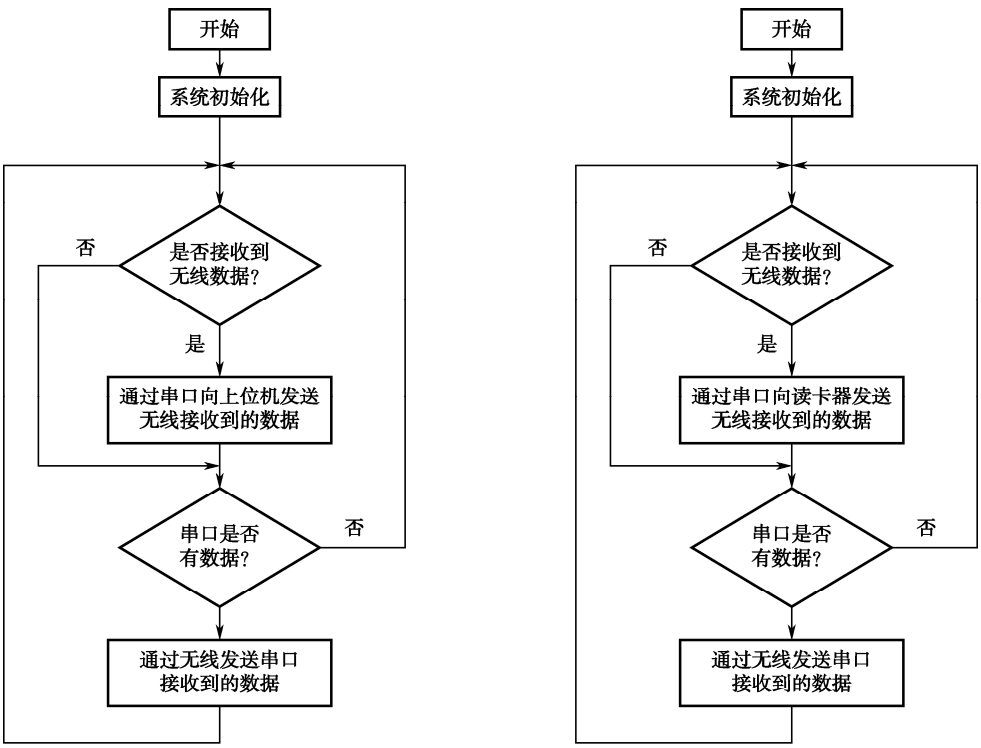
8.3 程序设计

1．上位机程序设计

RFID ISO 15693 读卡部分具体实验设计与第 3 章中的 HF13.56MHz ISO 15693 数据块的读写设计部分相同。

2．ZigBee 程序设计

RFID 实验箱 ZigBee 通信程序原理：上位机控制读卡器时，先向协调器的串口发出数据，协调器接收到串口数据后，再通过无线发送出去；然后再由子节点无线接收数据，最后子节点再通过串口把数据发送给相关读卡器。读卡器返回的数据也会按照原路径返回给上位机。程序执行流程图如图 8-6 所示。



（a）协调器通信部分程序执行流程　　（b）子节点通信部分程序执行流程

图 8-6　程序执行流程图

以下为相对应的具体程序解析。

（1）协调器串口读取程序：

```
static void rcBK(uint8 port,uint8 event)
{
  rxlen=Hal_UART_RxBufLen(0);     //接收缓冲区数据长度，单位为字节
  if(rxlen)
  {
    HalUARTRead(0,uartbuf,rxlen);   //从串口读取数据放在 uartbuf 缓冲区中
    GenericApp_SendTheMessage(uart_mode,uartbuf,rxlen);
```

```
    }
}
```

（2）协调器无线收发程序：

```
//无线数据接收函数
void GenericApp_MessageMSGCB(afIncomingMSGPacket_t* pkt)
{
  HalUARTWrite(0,pkt->cmd.Data,pkt->cmd.DataLength);    //写串口函数
}
//无线数据发送函数
void GenericApp_SendTheMessage(uint16 cID,uint8* buf,uint8 len)
{
  afAddrType_t my_DstAddr;
  my_DstAddr.addrMode=(afAddrMode_t)AddrBroadcast;
  my_DstAddr.endPoint=GENERICAPP_ENDPOINT;
  my_DstAddr.addr.shortAddr=0xFFFF;
  AF_DataRequest(&my_DstAddr,
                 &GenericApp_epDesc,
                 cID,
                 len,
                 buf,
                 &GenericApp_TransID,
                 AF_DISCV_ROUTE,
                 AF_DEFAULT_RADIUS);
}
```

（3）子节点无线收发程序：

```
//子节点无线接收函数
void GenericApp_MessageMSGCB(afIncomingMSGPacket_t *pkt)
{
  halUARTCfg_t uartConfig;      //该结构体变量是实现串口的配置
  switch(pkt->clusterId)
  {
    case uart_mode:       HalUARTWrite(0,pkt->cmd.Data,pkt->cmd.DataLength);
                          break;
    case set_uart_baud:   uartConfig.baudRate = *(pkt->cmd.Data);
                          uartConfig.configured=TRUE;
                          //uartConfig.baudRate=HAL_UART_BR_9600;
                          //uartConfig.baudRate=HAL_UART_BR_115200;    //波特率
                          uartConfig.flowControl=FALSE;                //流控制
                          uartConfig.callBackFunc =rcBK;
                          switch(uartConfig.baudRate)
                          {
                            case HAL_UART_BR_9600:   LED1_OPEN;
                                                     LED2_CLOSE;
                                                     LED3_CLOSE;
                                                     break;
```

```
                                case HAL_UART_BR_57600:   LED1_CLOSE;
                                                          LED2_OPEN;
                                                          LED3_CLOSE;
                                                          break;
                                case HAL_UART_BR_115200:  LED1_CLOSE;
                                                          LED2_CLOSE;
                                                          LED3_OPEN;
                                                          break;
                            }
                            HalUARTOpen(0,&uartConfig);                    //串口打开
                            break;
    }
}
//子节点无线发送函数
void GenericApp_SendTheMessage(void)
{
    afAddrType_t my_DstAddr;
    //发送模式：可选 单播、广播、多播方式，这里选 Addr16Bit 表单播
    my_DstAddr.addrMode=(afAddrMode_t)Addr16Bit;
    my_DstAddr.endPoint=GENERICAPP_ENDPOINT;      //初始化端口函数
    my_DstAddr.addr.shortAddr=0x0000;             //标志目的地址节点的网络地址
    //下面是数据发送
    AF_DataRequest(&my_DstAddr,
                   &GenericApp_epDesc,
                   GENERICAPP_CLUSTERID,
                   rxlen,
                   uartbuf,
                   &GenericApp_TransID,
                   AF_DISCV_ROUTE,
                   AF_DEFAULT_RADIUS);
}
```

（4）子节点串口读取程序：

```
static void rcBK(uint8 port,uint8 event)
{
  rxlen=Hal_UART_RxBufLen(0);     //接收缓冲区数据长度，单位为字节
  if(rxlen)
  {
    HalUARTRead(0,uartbuf,rxlen);   //从串口读取数据放在 uartbuf 缓冲区中
    GenericApp_SendTheMessage();
  }
}
```